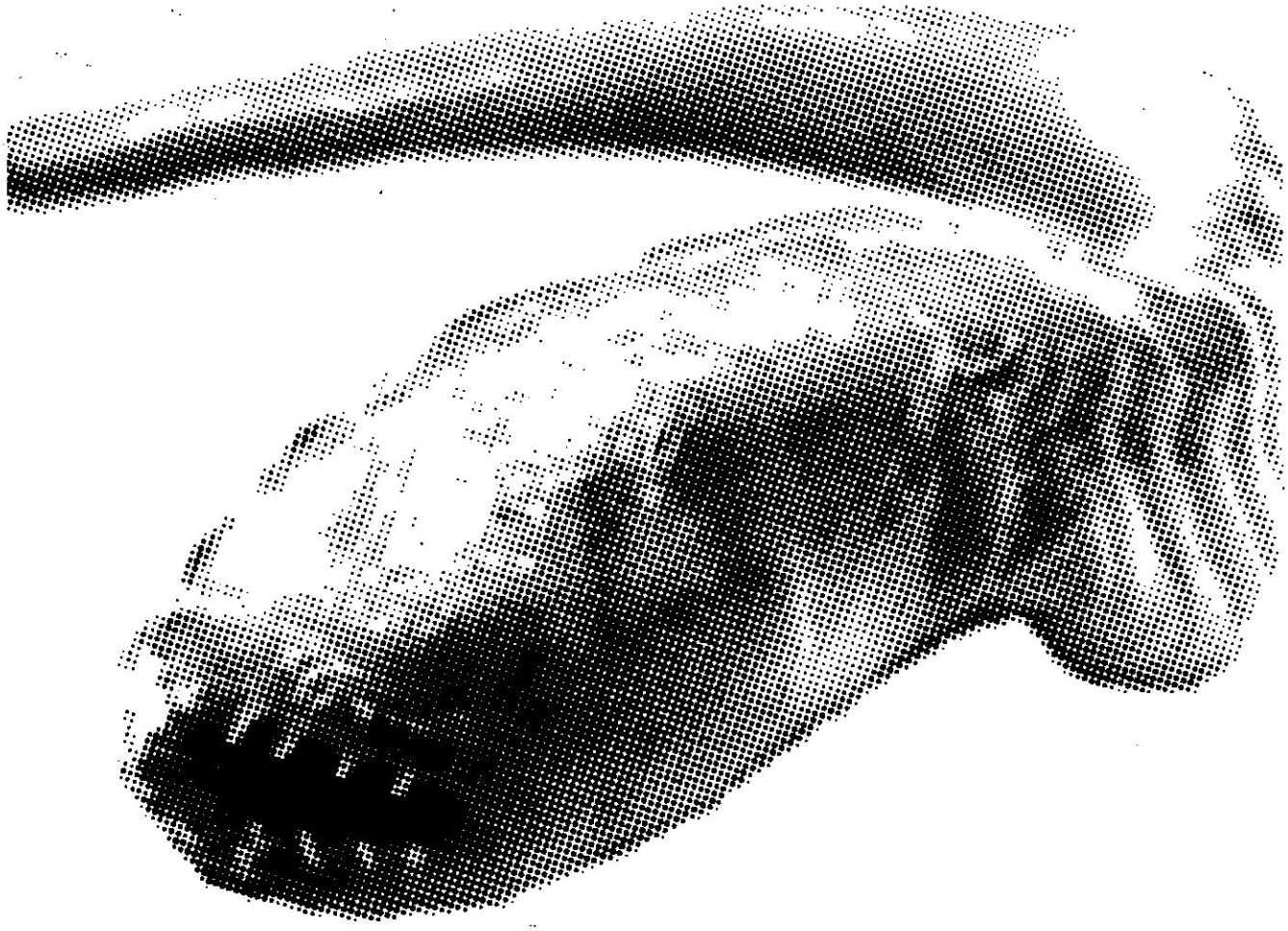
2002 spring



百萬石

for fresher edition

新入生歓迎の言葉

MMA 部長 佐野 浩司 (sano@mma.club.uec.ac.jp)

2001 年 4 月

MMA は Microcomputer Making Association の略で、昔は CPU を買ってきて半田づけなどをして独自のコンピューターを作っていました。現在ではコンピューター、主に PC 上で動く UNIX のサークルになっています。サークルとしてまとまって何かをするということは少ないので活動内容を一言で表すのは難しいのですが、プログラムを書いている人もいれば、ネットワークを運用している人もいますし、また PC を自作している人もいます。ようするにコンピューターを使って何かすることが好きな人達の集まりですね。

去年は ACM の主催する国際大学プログラミングコンテスト (ICPC) に 1 チームが参加し、国内予選を勝ち抜きアジア地区予選に進出しました。また夏合宿も毎年行っていますし、11 月の学園祭 (調布祭) では部員が各々の活動内容の展示を行い、ジャンク屋も開かれていました。

MMA 部室内では数台のマシンが稼働中で、ネットワークにも接続されています。

MMA は、縁と環境を提供します。MMA に入部し同好の士を見つけてはいかがでしょうか。

ICPC第一次中間報告

ICPC対策委員長 吉田 康弘 (niku@mma.club.uec.ac.jp)

2001年11月

概要

M.M.A.一年の吉田康浩です。この記事では、先頃、僕が出場した国際プログラミングコンテストの詳細をみなさんに紹介します。この記事を読んで、少しでもコンテストに興味を持った部員の方はすぐにM.M.A.-ICPC圧勝プロジェクトに参加しましょう。そして、来年こそ晴れ晴れとした顔で羽田空港に戻って来ましょう。最後に、今回僕らのために部費を使うことを許可してくださった部員の方々にあらためてお礼を申し上げます。

1 ?そもそもICPCってなんですの?

ICPC(International Collegiate Programing Contest)とはACMが主催する国際大学対抗プログラミングコンテストのことである。このコンテストはとても由緒のあるものらしく、たくさんの有名企業がスポンサーについていた。また国際コンテストであるから、公用語は英語である。(ここ重要!)そのため、大会に出場する時は辞書は必需アイテムである。コンテストは、出題者に与えられた問題をいかに早く解くかを競う、という形式である。一般的な、「みんなで作ったソフトを持ち寄って、だれのソフトが一番インテリジェントかにゃー」ということを競う、BASICマガジン的な、マターリ大会ではない事に注意しよう。そのため、コンテスト会場には、張り詰めた空気が漂っている(一部例外アリ)。大会の解答時間中は、コンパイルに使用する計算機以外の一切の電子機器の使用を禁じている。電子辞書はもちろん、携帯電話を時計がわりに使うのも禁止である。(じゃあ、電磁はえたたきは?)また、計算機中にあるマニュアル、大会の問題ウェブページーを除く一切のファイル、ウェブページの閲覧を禁止されている。問題は言語依存のシステムを使うのではなく、基本的な命令だけで解けるようになっている。問題は英語で書かれており、日本人の我々としては、まず問題を間違いの無いように理解することが大切である。回答時間は大変短い。(以下参照)

問題の形式は次のようになっている。(問題文超要訳)

アジア地区予選

Problem A

"Starship Hakodate-maru"

Input:ssh.txt

次の式を満たすような最大の正整数Zをもとめよ。

$$Z = n \times n \times n + m(m+1)(m+2) \qquad (n, m \text{ は正整数})$$

ただし、Zの上限はSample Inputで与えられる。

(本来はここに、この問題についての逸話がついている。この場合は、この数値が、「宇宙船函館丸の燃料タンクに積める球状の燃料の個数であって...。」といった、大会主催者のおちゃめなおはなしがついていた。大会の常連の話を聞くと、「役に立たないから読まない」そうだ。来年出る選手はよく過去問を解いて、問題の重要な部分だけを読むようにしよう。間違っても、"Once upon a time..."などを読んではいけない!!)

Input

入力は最大1024個の正整数で与えられる。入力されるテキストファイルの行にはそれぞれ一行に付き1つの正整数を含み、行が0を含めば、そこがファイルの終りである。また、それぞれの正整数は最大で151200である。

(ここで、問題に出てくる変数や、与えられる入力の上限などが記載される。ここがコーディングする上でもっとも重要になる部分である。ここは穴があくほど読んで、日本語で内容のメモを取っておこう。)

Output

一行にひとつずつ解答の正整数を出力せよ。ただし、順番は与えられた順とする。
（ここで答えの出力の形を指定される。この要求にあった、標準出力であるものだけが正解である。つまり、正解は1つしかないのだ。）

Sample Input

```
100
64
50
20
151200
0
```

（これが、ここに書いてある通りテキストで与えられる。ファイルの名前は問題の題の下に書いてある、"ssh.txt"である。この事実を見落としていたため、僕達は会場で混乱することになる。これもすべて英語力の不足がいけないのである。）

Sample Output

```
99
64
47
20
151200
```

（プログラムを作成して、このような標準出力を得れば正解である。たとえ答えがあっていても、上のような形でなければだめなのだ。）

この大会は企業がバックに付いていることもあり、「想像力」「個性」よりも、むしろ「スピード」（要求されたことをいかに早くやるか）に重点が置かれているようだ。いくら文部科学省が「個性」を強調しても、時代はやはりスピードなのか…。

大会は次のような順序で行われる。

1. 国内予選
2. アジア地区予選
3. 世界大会（決勝）

国内予選は"日本の中の大学"からアジア地区大会に出場するチームを選出するものである。これは、インターネットを活用して、ネットワークを通して行われる。成績優秀チームは、アジア地区の他のエリアでの予選に送り込まれる。

アジア地区予選はアジアのなかの"日本エリア"で行われるものである。そのため、上記のシステムにより、若干、外国チームがいる。ここで優勝したチームだけが世界大会に出場できる。

「毎年恒例だが、日本チームは"日本エリア"予選で優勝したことがないそうだ。」

と、大会の司会をやっていた異人さんはいっていた。でも、心配ご無用。来年は、電気通信大学チームが晴れて優勝し、世界大会に出場する予定でございます。

世界大会（決勝）は、ハワイであります。詳細は今度行って確認してくる予定。おみやげはマカダミアンナッツチョコという方向で。

2 大会の様子

選手団（敬称略）

- コーチ
  - TATEOKA Takamichi (tree)

- 選手
  - SATOU Takashi (ikidou)
  - Sakurai Kenji (sakura)
  - KIKUCHI Kouji (xmoe)
  - YOSHIDA Yasuhiro (niku)

2.1 国内予選

日時: 10 月 5 日午後 6:00 ー午後 8:00（2 時間）

場所: 電通大 西 1-306 楯岡先生の研究室

そもそも僕がこの大会にでるきっかけとなったのは、正式メンバーである xmoe 氏が、絶対行かなければならない授業に行かなければならず、メンバーに空きができたためである。本来 xmoe 氏が出場という栄誉？を受けるはずだったのに、僕が出れることになり、ちょっと嬉しかったりした。

さて、国内予選で、僕達は下記のような計算機環境を使用した。

OS: FreeBSD4.4-Release

Compiler:gcc

大会が始まる前、解答のテスト送信などがあって、そこで解答をメールで送る練習ができるようになっている。
解答方式はこうである。

- 大会が始まった瞬間に ICPC のウェブページに問題が掲載され、見れるようになる。
- それを持ってきて、みんなで解く。（三人集まりゃ文殊の知恵）
- 解けたっ！と思ったら ICPC のウェブページから問題となっている Input 用のテキストを持ってくる。
- それを実際に入力して実行。（僕達は標準入力から読み込んだ）
- 結果をメールで大会本部に送信。
- 正解ならばそれを知らせるメールが本部から返ってくる。
- 正解したならばそのソースを大会に送りつける。

一連の動きを２、３回繰り返して、正しいかどうか確認。というわけだ。
僕達の作戦はこうだった。

- キータイプの早い ikidou がコーディングを担当。
- そして、sakura と niku が英訳＋アルゴリズムを担当する。
- ikidou は他の二人が一問目を考えている間に、SampleInput の読み込み部分などの全
- プログラム共通のソース（前もって考えてあり、紙に印刷）を通常の三倍の速度で打ちこむ。

当初の目標は全問制覇だった。
始まってみて、これがなかなか大変だという事がわかった。問題が英語なのが大変だった。自分の読んだ内容で問題を考えて本当によいのか、という不安が常に付きまとったのだ。そのため、しっかり英語を読んでしまう。すると、問題を解くのが遅くなるわけだ。
結構練習していったつもりだったのだが、結局一問しか解けなかった。最初、簡単な第一問が解けたのだが、その後文字列整形で結構時間をとられた。
でも、なんとかアジア地区予選にでることができたのだった。

結論:
試合において運は重要なファクターです。試合前にみんなでお祈りをしましょう。
「神のご加護がありますように」ってね。

2.2 アジア地区予選

日時: 11 月 10,11 日 (本大会：11 日 午前 9:30-午後 2:00)

場所: はこだて未来大学

11 月 10 日

往路あった数々のいざこざにめげる事なく、僕ら選手団は函館空港に到着した。さっそく僕達は今日泊まるホテルに行くために、バスに乗り込んだ。バスのなかで話される話題はもちろんコンテストのこと、ではなく今日の昼飯のことだった。
ホテルに着いたとき、最初、僕は自分の泊まるホテルを間違えてしまった。なぜなら、遠くに見える、おっきな、きれいなホテルにたくさんの同業者の方々が入っていくのを見たからだ。少し経って、やっと僕達が泊まるホテルは、さっきのホテルの横の、コンパクトでブリチーなホテルだという事に気づいた。僕はビジネスホテルに泊まるのが初めてだったので、少し嬉しかった（嘘）。これが世の中の厳しさだということを、ICPC は教えてくれたに違いない。僕はコンテストを頑張ろう、という気がしてきた。

来年に向けて:
「予選でいい成績 ならば 大きなホテル。」「大きなホテルがいいホテルとは限らない」
この命題が真であるのか偽なのかを証明する。

午後一時頃、バスで会場に行き、はこだて未来大学に着いたとき、大学の綺麗さに、僕は驚いた。一口でいうと、「ターミナルドグマ」なのだ。（説明でわからない人は、はこだて未来大学のウェブページを見てくださいな。）会場は学内で一番広い部屋？で、天井までの高さが優に四階立分。電通大体育館よりも広いだろう。空調が行き届いているのが印象的だ。会場には一チームに対して一台のパソコンと、四つのハイテク風机が与えられている。
会場では次の作業をする事になっている。

- 選手の登録をする

- 大会での環境に慣れる＆使い方マスター
- JAVA コンテスト

大会での環境に慣れる＆使い方マスター

僕達はあまり大会の環境での練習を積んでおかなかったので、ここで苦労することになった。
大会では次のような環境を使うことになっている。

Hardware: IBM IntelliStation E Pro

OS: Windows NT 4.0 SP5 (Japanese version)

Compiler: VisualAge シリーズ

CPU: PentiumIII 500MHz

メモリ:128MB

ハードディスク: （コンテスト問題を解くのに十分な容量）

ディスプレイ: Sony CPD-L200(15 インチ液晶モニタ、1024x768)

マウス: 3 ボタンマウス

キーボード: 普通の日本語キーボード

とりあえず僕達は、併設された勝手に食物を食べていいゾーンにいき、餌をむさぼった。(これで結構時間のロスをした。)
お試し問題の冊子が配られた。アジア地区予選ではプリンターは全員で二台を共有する形である。だが、問題の紙だけは配られたので、当初予想された問題を印刷するためにプリンターを使う必要が無くなった。さて、大会でのマシンの使い方の詳細が書いてある冊子（英語）を読んでいて、僕らは予選との大きな違いに気付いた。大会では、SampleInput をプログラムに食わすときに、標準入力を使うことが禁止されていたのだ。このような制限がつくと、プログラム中でファイルを読み出す機構をもうけなければならない。こんなこと、M.M.A. 部員には朝飯前だと思われたが...。
さて、お試し問題を解いていたときに問題は発生した。

「プログラム中で読み込むファイルの名前はどうすんの？」

標準入力でない以上、読み込むファイルの名前を決め打ちで打たねばならない。後天的に指定させることもできようが、審査マシーンがこんな器用なことできるわけがない。そんなわけで、勇気を出して係の人（日本人）に英語で質問をすることにした。ネイティブには絶対通じないであろうたどたどしい英語の説明の後、係の人は先ほど配られた冊子のある一点を指さした。そこには英語で

「ただし、Input ファイルの名前は、問題の題名の下に書いてあるものを使う。」

と、記されていた。おそるべし、M.M.A. の英語力 !!
次に問題となったのはデバックの方法である。今まで見たことのないインターフェイスを持つ VisualAge シリーズを前にして、僕らは悪戦苦闘をしていた。何より苦戦したのはデバック時、エラーメッセージがどこに出るのか、である。普通に使うと、エラーが出ても「ぴっ！」というビープ音が出るだけなのだ。少し経って、僕等はやっとのことでウインドウ上のタグを押す事で、メッセージの表示されている場所にありついたのだった。

教訓:

- 説明冊子を穴の開く程読もう。

- 指定されたプログラミング環境での練習をもっとする。特に、デバックの方法は必修である。
- 英語での質問のテンプレートを作っておこう。

  例)
  - 「この hoge の意味を教えてください。」
  - 「問題が全て終ったら、他の参加チームを手伝ってもよいのですか？」

- 英語をもっと勉強しよう。

JAVA コンテスト

次は、恐怖の JAVA 大会である。
この大会は、将来性の高い言語であると企業側が主張する JAVA を普及させるための取り組みの一環である。そのため、問題も本大会に比べると、(JAVA を知るものにとっては) 比較的簡単なものになっている。大会前、JAVA は niku の担当という事になっていた。しかし、niku は、何を勘違いしていたのか、JAVA アプレットの作り方を学んでいたのだ。しかし、大会で出たのは本大会の問題の JAVA 版であった。この手の問題を解く時に、まず始めにやらねばならぬ事は、ファイルの読み込みである。この問題は、歯を踊らせる事しか頭に無かった niku にとっては少し荷が重かった。(だからといって niku のアプレット技術がそんなにあったわけではない。) しかし、ここで強力なすけっとが！sakura さんがかなり JAVA をマスターしていたのだ。さすが部長 !! ありがとう、sakura さん！
しかし結局、Input ファイルの読み込みすら出来ぬまま大会は終ってしまった。残念だ...。
この日は 午後六時頃でホテルに戻った。

練習日の教訓:

- 餌食っている暇があるんだったら操作方法をマスターする。
- JAVA を覚えよう。

11 月 11 日

本大会

この日は、朝八時半からバスで会場に向かった。前の日と同じように会場で登録を済ませた後、会場にはいり、スタンバイした。昨日と同じく、パソコンが一台に机四つの構成だ。ここで注意したいのは、荷物は自分達のチームの机の横に置いておけない事だ。そのため、必要なものはあらかじめ出しておく必要がある。今回、僕は高校数学公式集を出すのを忘れていた。それがそのまま出ている問題があったので、自力で計算するのがめんどくさかった。
今回の大会では、問題を一問解くごとに一つ風船をくれるようだ。つまり、解いた問題の個数が他のチームに知れ渡るという事である。
午前九時半にゲームが開始された。予め紙で配ってあった問題の冊子を急いでみんなに渡す。(問題の詳細は以下のウェブページ[1]から持ってきてください。)
まず僕は一番から読み始めた。sakura さんは二番を解き始めたようだ。一番は非常に簡単な問題だ。多分、主催者側がお土産の風船を与えるために作った問題であろう。使う式も、インプットするファイルの作りも簡単だ。僕としては、この問題は鶴亀算的手法でときたかった。
一番の方法を ikidou さんに伝えた後、僕は三番を解き始めた。三番は過去問で解けなかった図形問題のようだ。過去問と同じような面積問題のようなので、出来るとふんで、解く事にした。途中、式の整理に手間取ったが、一応出来たようだ。
さっき出来た一番の問題の風船が来た。やっぱりなんかできると嬉しい。風船一個、ゲットだぜ、とかつぶやいてみた。

[1] http://www.fun.ac.jp/icpc/index-j.html

横をみると、ikidou さんの手が止まっている。二番を見てみたら、どうも文章整形の要素が盛り込まれているようだ。文章整形は僕達はどうも苦手なようなので、僕はとりあえず方法が分かっている三番をやることを提案した。

しかし、その後の問題がなかなか解けなかった。とりあえず五番に挑戦して挫折した後、僕は sakura さんが考えていた七番を一緒に解くことにした。

風の噂で 7 番を考えている人が多いというので、二人の考えを書いておく。(まちがっていたらごめんなさい。)

sakura:
この問題にもゴールデンルールがあり、それを見つけ出せば問題は一瞬にして終る。

niku:
この問題はパソコンに解かせるものである以上、何らかの形でその計算能力を使わなければならない。この問題を解くにあたり計算能力を使う方法で考えられるのは、この問題をシュミレーションすることである。

そんな事を考えている内に早くもコンテストが終ってしまった。

「へぇ、四時間半ってこんなに短いんだー。(脱力)」

というのが素直な感想である。

結局、三番の問題はバグとりが終らず、解答できなかった。

結果 風船一個 (ProblemA) only

本大会での教訓:

- やっぱりコーディングは考えた人自身がやったほうがよい。
- そのためにメンバー全員がもっとスキルアップする。
- アルゴリズムをもっと系統立てて勉強する。今回の問題でも、半分くらいはアルゴリズム本に載っているものに似たものが多いようだ。よく考えたら、僕は今までアルゴリズムの勉強をした事が無い。
- 問題の傾向を研究する。JAVA コンテストにアプレットは出ない、とか。

コンテストが終ってから、少し立食パーティーがあって表彰式が行われました。やっぱり、自分の表彰されない表彰式には出るもんじゃないな。

パーティ中、コンテスト主催者側の、問題に対する総評がまとめられていました。来年はこれをしっかりメモしましょう。今年の分は僕が自分の Visor にメモしておいたのですが、帰りにそれ自体をトイレに落すというアクシデントのため、データーが失われてしまいました。残念です。

2.3 来年に向けて

来年のアジア地区予選は金沢でやるそうです。来年に向けて M.M.A. ではアルゴリズム勉強会などが催される予定です。部員のみなさんも、積極的にこのような集まりに参加して、自分の腕を磨きましょう。そして、「日本国に M.M.A. あり。」と言われるまでに成長するという方向でどうでしょうか。

3 あとがき

コンテストが終らないとこの記事が書けないのに、記事の〆切はそれより前。いま、頑張って書いているけど、もしこの記事が載ったら僕はとてもうれしい。結構書く事があってまとめるだけでも大変でした。しかも、結構手抜きをしてしまいました。あっちこっちに間違いがあったり、話のつじつまがあわなかったりするかもしれないけど、勘弁してください。

あぇ、学校に遅刻するー。

Problem C
Another Mine Sweeper

菊池 孝治 (xmoe@mma.club.uec.ac.jp)

2002 年 4 月

概 要

大志溢れる新入生のために、ICPC (International Collegiate Programming Contest) 関連企画として、本番の雰囲気さながらの仮想問題を提示し、それを考えてみる、という演習のようなことをしてみたいと思います。これを読んで(もしくは読む前から)、プログラミングをもっと楽しみたい!! と思った新入生諸君、今すぐ MMA に入部しよう!!

1 仮想問題

Prologue

西暦 2048 年、地球から遥かに離れた U-0xEC 星では、ハンガリアン派と反ハンガリアン派の宗教対立による戦争が、長年に渡って続いていました。

U-0xEC 星の circle 島一帯を実効支配する、反ハンガリアン派の MMA (Micro Megane Attackers) の隊長である sano 氏から、敵対勢力の地雷利用に対処することを命じられた、MMA 情報工作省 (夜間主コース) の xmoe 工作員は、隊長を前に一つの情報を報告していました。

> 「隊長、我が惑星から 256 光年の距離にある" 地球" という星では、" 計算機" という機械に、半ば強制的な形で、Windows というモノがバンドルされている模様です。Windows というモノには、"Mine Sweeper" と呼ばれる機能があるらしく、先日拉致した地球人によれば、これは" 地雷掃除機" を意味するということです。また、これを開発し販売した Bill Gates という生物がほぼ地球全域に渡る独占的支配を確立していることからも、Windows というモノが、なにか軍事的なモノであることが伺えます。」

sano 隊長は静かに答えました。

> 「うーむ、" 地球" という星では、既に地雷掃除機なるものが開発されているのか。よし、xmoe には、その Windows というモノの入手を命ずる。万一の場合の覚悟はできているな。」

xmoe 工作員は、緊張した面持ちで答えました。

> 「超時空船の自爆システムの動作はチェック済みです。しかし、一般漁船としての偽装工作も完全を期していますので、何としてでも任務を遂行して帰港致す所存であります。」

数年後

地球から見事に "計算機" と、バンドルされていた Windows というモノを入手してきた xmoe 工作員を待っていたのは、怒り狂った sano 隊長でした。

> 「"Mine Sweeper" という機能が、我々に" 地雷掃除をさせる" ものだとは、一体どうなってるんだ!!」

ということで、あなたに与えられた任務は、その名の通りの"Mine Sweeper" プログラムを作成することなのです。

Input

入力は、初期マップの数を表す n という正の整数 ($n \leq 1000$) と、その次の行からの n 個の初期マップで与えられます。

n
初期マップ1
初期マップ2
初期マップ3
...
初期マップn

それぞれの初期マップは、下記のような書式で書かれています。

```
title
w h
pppppppppp
pppppppppp
pppppppppp
pppppppppp
pppppppppp
```

最初の行は、マップのタイトル(改行を含まない 100 文字以下の文字列)です。

その次の行の、w と h は、それぞれ 1 以上 100 以下の整数であり、初期マップの横方向と縦方向の大きさ(一つの要素を単位とします)を表します。

その後、改行に続いて、w 文字からなる h 行の、マップ上の各要素の情報を表す文字が続きます。それぞれの文字は、0 から 8 の間の整数、もしくはハイフン '-' であり、

- 整数の場合は、その数字は、その要素の周囲 8 要素(上下左右と左上、右上、右下、左下の要素)のうち、地雷が埋められている要素の数を表します。整数が示されている要素自体には、地雷は埋められていません。
- ハイフン '-' の場合は、その要素に地雷が埋められているかどうかが不明なことを表します。

あなたが作成するプログラムに要求されることは、与えられた初期マップをもとに、"確実に地雷が埋められている要素" と、"確実に地雷が埋められていない要素" を全て求めることです。また、与えられた初期マップに矛盾が生じている場合も、それを検出しなければなりません。

Output

入力された初期マップそれぞれについて、マップのタイトルを出力し、次の行からは w 文字からなる h 行の結果マップを出力して下さい。

結果マップは、下記のような文字から構成されていなければなりません。

シャープ '#'
その要素には、確実に地雷が埋められていないことを表します。

アットマーク '@'
その要素には、確実に地雷が埋められていることを表します。

ハイフン '-'
初期マップからは、その要素に地雷が埋められているのか、埋められていないのかが確定不能なことを表します。

また、初期マップに矛盾が生じている場合は、結果マップの代わりに、

```
Contradiction!!
```

という行を出力して下さい。
マップごとの出力の間には、空行を一行、出力することが求められます。

Sample Input

```
6
Practice-1
3 3
---
-0-
---
Practice-2
3 3
---
-8-
---
Practice-3
3 3
3--
---
--1
Practice-4
3 3
4--
---
---
Practice-5
3 3
2-1
---
---
Practice-6
3 3
1-1
-3-
1-1
```

Sample Output

```
Practice-1
###
###
###

Practice-2
@@@
@#@
@@@

Practice-3
#@-
@@#
-##

Practice-4
Contradiction!!

Practice-5
#-#
@-#
---

Practice-6
Contradiction!!
```

2 どうやって解きましょう

2.1 早速

ということで、解いてみます。

と、書き出そうかと思いましたが、ちょっと待ってみましょう。
というのは、僕自身、これから書こうと思っている解答に辿り着くまで、この問題をぼんやりと考え始めてから一週間くらいかかってまして (実際、そのへんの回転がへぼいです)、その間の思考内容の紆余曲折を逐一解説しながら、読者の方に追体験してもらえたら素晴らしいと、(少なくとも僕は) 思うのですが、紙面のスペースとその他諸々(主に時間と作文能力) の関係上、それは難しい状況です。

なので、プログラミング大好きで、このパズルにも興味を持った新入生の方は、是非ここで読むのを一時停止して、考えてみて下さい。考えた後に解答例を読めば、

この記事を書いてる xmoe っていう自称工作員は、こんなにオバカだぜ。

とかいう優越感を味わえるかもしれません (そのときは、オバカな筆者にも、あなたの優秀な解答を教えて下さいね!!)

2.2 ウォーミングアップ

本格的に問題に取り掛かる前に、これからの解説で使う用語を定義しておきます。

まず、毎回、「ある要素の上下左右、右上、右下、左下、左上の要素」というのはめんどくさいので、これを

ある要素のムーア近傍の要素

と呼ことにしましょう。

次に、マップ上の各要素の分類です。作成するプログラム上では、マップ上の全ての要素を、下記のうちの一つの状態を持つものとして扱うことにしましょう。

未判定要素
最終的には、他の 3 つの要素のうちの 1 つに分類される要素であるが、その判定を行っていない要素。

地雷要素
地雷が埋まっている要素。

空要素
地雷が埋まっていない要素 (数字のついている要素も、これに該当します。)

独立要素
その要素のムーア近傍に数字が示された要素がなく、要素自体にも数字が示されていない要素。

上の分類のうち、「空要素」については、さらに二つに分類することができます。

数字なし空要素
空要素のうち、その要素に数字が示されていないもの。

数字あり空要素 (または数字要素)
空要素のうち、その要素に数字が示されているもの。

3 本格的な (?) 解説 そのいち

3.1 方針

ということで、解説に入ります。できるだけ、筆者の考えを、ゆっくりと正確に書いていくつもりですが、至らぬ点は、読者の読解力と忍耐力を存分に発揮し、何度も読み返すことでカバーしてあげて下さい。

3.2 入力

問題文に示されたような入力から、マップ上のそれぞれの要素の状態を、適当な配列に入れていきましょう。初期マップの要素それぞれについて、

- 数字が示されている要素については、数字要素
- ハイフンである要素については、未判定要素

という形で、配列に記憶していきます。

3.3 独立要素であることの判定

次に、さきほど記憶したマップ上の全ての未判定要素について、独立要素の判定をしてしまいます。

ムーア近傍に数字要素がない未判定要素について、なぜ「独立要素」という判定をしてしまうのかというと、そのような要素については、

- 最終的に、地雷が埋まっているかどうかの確定ができない。
- その要素に地雷が埋まっている、または埋まっていない、という仮定が、他の要素に関する判定に影響を与えることがない。

といった特徴を持つからです。(ここで厳密な証明をすべきなかもしれませんが、筆者の手に負えませんので、証明ができる方がいれば、教えて下さい。)

また、これからプログラム上で、未判定要素を判定していった時に、ある未判定要素を数字要素として判定する、ということはしませんので、未判定要素の判定が進むと同時に、独立要素である要素が他の要素になったり、その逆になるようなことはありません。ですので、初期マップを読み込むと同時に、独立要素であることの判定をしてしまいます。

3.4 地雷要素か空要素であることの静的判定

さて、最終的な答えを得るための足がかりとして、未判定要素のうち、そのムーア近傍にある数字要素[1]と、その数字要素のムーア近傍にある要素みるだけで、地雷要素か空要素のどちらかであるかを判定できる[2]ものについての判定方法を述べてゆきます。

少し考えてみると、次のような規則があることが発見できると思います。

地雷要素であることを静的に判定する規則

ある数字要素に注目したときに、"その数字から、その数字要素のムーア近傍にある地雷要素の数を引いたもの"が、"その要素のムーア近傍にある未判定要素の数"と等しいとき、その数字要素のムーア近傍にある未判定要素については、すべて地雷要素であると判定してしまってよい。

数字なし空要素であることを静的に判定する規則

ある数字要素に注目したときに、その数字と、その数字要素のムーア近傍にある地雷要素の数が等しければ、その数字要素のムーア近傍にある未判定要素については、すべて数字なし空要素だということを判定してしまってよい。

なにやら良く分からない日本語ですね。簡単な話、Sample Input 中の 1,2,3 番目みたいな問題については、パッと見で、数字のある要素の周辺の、地雷が埋まっている要素と、地雷が埋まってない要素を判定できるじゃん、ということを言っているわけです。

3.5 数字要素に関する静的矛盾チェック

次に、矛盾チェックです。

問題文には、入力初期マップの中に、矛盾をしているようなものがあり、それを検出しなくてはならない、ということが書いてあります。また、後に述べる動的な探策においても、矛盾してしまっているマップの探策はしないことが必要[3]ですので、マップの矛盾をチェックをする方法を考えなくてはなりません。

天与の法則的な書き方で申し訳ないのですが、これについても、少し考えてみると、次のような規則で検出できることを見いだせると思います。

---

[1] 最初に、数字要素のムーア近傍にない (=ムーア近傍に数字要素がない) 要素については、独立要素として判定してしまいましたので、未判定要素のムーア近傍については、最低一つの数字要素が存在することになります。

[2] この判定には、「問題文に矛盾がない限り」という前提条件がつきます。

[3] バンバン探索して、矛盾チェックは最後に行う、という方法もあります。また、前に述べた静的判定をせずに、ばしばし探索する、という方法もあります。本稿ではその辺の話を省略してしまっていますので、何かあったら筆者に直接ふっかけてみてください。

地雷要素が多すぎる矛盾規則

ある数字要素に注目したときに、その数字よりも、"その数字要素のムーア近傍にある地雷要素の数"のほうが大きい場合、矛盾が生じている。

地雷要素と未判定要素が少なすぎる矛盾規則

ある数字要素に注目したときに、その数字よりも、"その数字要素のムーア近傍にある地雷要素の数と、その数字要素のムーア近傍にある未判定要素の数の和"のほうが小さい場合、矛盾が生じている。

たとえば、Sample Input 中の 4 番目のマップについては、「地雷要素と未判定要素が少なすぎる矛盾」により、矛盾が生じていることを検出することができます。

3.6 静的に解いてみる

–Sample Input の 1,2,3,4 番目の初期マップについては、これまでに示した方法のみを使えば、Sample Output と同じような出力を得ることができます(人間がやっても、大した時間はかからないと思いますので、ぜひ試してみて下さい。)

しかし、5 番目や 6 番目のような初期マップについては、これだけで解くことはできません。これらの問題を解くには、「静的判定では判定できない未判定要素について、仮の判定をすることを繰り返し、最終的に矛盾の生じないような地雷配置を全て求め、それに基づいて出力を行う」ことが必要になります[4]。

4 解説 そのに

4.1 方針

Sample Input の 5 番目に示されるように、静的な方法だけでは判定できないような未判定要素があり、また、Sample Input の 6 番目に示されるように、静的な方法だけでは検出できないような矛盾があることが分かりましたので、前節の最後で述べたような、

> 静的判定では判定できない未判定要素について、仮の判定をすることを繰り返し、最終的に矛盾の生じないような地雷配置を全て求め、それに基づいて出力を行う

というようなプログラムを書く方法を考えてみます。

その前に、「スタック」と「再帰」という概念の解説から。。。

4.2 スタック

スタックというのは、コンピュータでは盛んに利用される、基本的なデータ構造の一つで、「最後に積んだものが最初に取りだされる」ようなデータの保存法をいいます。

机の上に本を積んでおくような状態を想像してみて下さい。一番上に更に本を積むこともできますし、本が一冊以上あれば、そこから本を取りだすことができます。本を取りだすときに、一番上の本を取りだせば、それは一番新しく積まれた本なわけです。

数字で例を挙げると、

- あるスタックに、5 を積んで、取りだすと 5 が得られます。
- あるスタックに、5 を積んで、更に 4 を積んで、取りだすと、4 が得られます。更に取りだすと 5 が得られます。

なんとなくでも分かっていただけたでしょうか。この記事を読むためには、これくらいの知識で事足りると思います。

[4] と筆者は思うのですが、そんなことしなくてもできるよ、という方は、すぐにでも教えて下さい。お願いします。

4.3 再帰

再帰というのは、プログラム(やアルゴリズムやデータ構造やその他諸々)を記述する上での手法の一つで、「何かを記述する上で、それ自身を用いる」ようなこと(?)をいいます。

よく出される例として、"階乗"の求め方があります。

普通に n (自然数) の階乗を求める場合、

> 1 から n までの自然数を、順番に掛けていく

という方法がとられますが、

> n が 1 であれば、n の階乗は 1 である。n が 1 でなければ、n の階乗は、$(n-1)$ の階乗に n を掛けたものである。

という形でも階乗を求めることができると思います。、"階乗を求める"という作業の中に、階乗を求めるという手順があるわけで、このようなものを"再帰"といいます

スタックや再帰についてもっと詳しく知りたい方は、図書館で、「アルゴリズムとデータ構造」に関する本を一冊読んでみて下さい。きちんとした本であれば、これらの事柄について、詳しく書かれていると思います。

4.4 仮の判定とバックトラッキング

ということで、ここから、本稿の核心である、「静的判定では判定できない未判定要素について、仮の判定をすることを繰り返し、最終的に矛盾の生じないような地雷配置を全て求める」方法を解説していきたいと思います。

という予定だったのですが、印刷まであと 1 時間ありませんので、その手順をいきなり紹介してしまいます。

あるマップについて、探索をするとは

1. マップ上の全ての未判定要素について、地雷要素か空要素であることの静的判定を行う。(判定できない要素については、未判定要素のままにしておく)
2. 数字要素に関する静的矛盾チェックを行う。
   - ここで矛盾が検出された場合、そのマップについての探索は終了とする
3. マップ上の未判定要素を探し、
   - 未判定要素がなければ、現在のマップは、最終的に矛盾の生じないような地雷配置をあらわしている。
   - 未判定要素がみつかれば、その未判定要素に注目し、
     (a) マップの状態をスタックに積む。
     (b) 注目している未判定要素を地雷要素として(仮に)判定する。
     (c) そのマップについての探索を行う。
     (d) マップをスタックから取り出す。
     (e) マップの状態をスタックに積む。
     (f) 注目している未判定要素を空要素として(仮に)判定する。
     (g) そのマップについての探索を行う。
     (h) マップをスタックから取り出す。
4. そのマップについての探索は終了とする。

詳しい解説ができないのは申し訳ないのですが、上記の手順を Sample Input 中の 5 番目や 6 番目に適用してみて、どのような形で処理が進むのかを考えてみると、なんとなく理解できてくるのではないかと思います。

また、上記の手順中には、「マップの状態をスタックに積む/取り出す」という操作がありますが、これは、

ある時点でのマップの状態をマークしておき、後でその状態を復元する

ことができればよいわけで、バカ正直にマップをスタックに積んだりしなくても可能です。これは、読者への課題としておきます (たとえば、囲碁であれば、棋譜さえ取っておけば、局面を n 手前の状態に戻す、ということは簡単にできますね。)

ちなみに、ここで述べたようなアルゴリズムは、一般的に" バックトラッキング" 又は" 深さ優先探索" と呼ばれるもので、この種の、パズルに関する問題では、激しく頻出な手法です。

ICPC でも、このようなパズル問題はよく出題されますので、結果として" バックトラッキング" は、ICPC に参加するにあたって、必ず身につけておきたいアルゴリズムの一つとなっています。

4.5 最終的な出力

さて、" 最終的に矛盾の生じないような地雷配置を全て求め" る方法は分かりました。

しかし、問題は、" 各要素が地雷要素であるか、空要素であるか、または判定不能な要素であるかを表す、一つの結果マップ" を出力することを要求していますので、探策により求めた、可能性のある地雷配置 (複数形) から、このような単一のマップを作成する方法を考えなくてはいけません。

筆者が一番簡単だと思う方法は下記のようなものです。

まず、初期マップを入力すると同時に、マップ記憶用の配列と同じ要素数を持つ、結果保存用の配列を二つ用意します。

二つの配列は、それぞれ

- "地雷要素である可能性" があるかないか (地雷可能性マップ)
- "空要素である可能性" があるかないか (空可能性マップ)

を表すことにし、最初は、全ての要素を、「可能性なし」に初期化しておきます。

次に、バックトラッキングによる探策を行い、" 最終的に矛盾の生じないような地雷配置" が求まるごとに、マップ記憶用配列の各要素を基に、下記のような操作を行っていきます。

- 記憶用配列の要素が、" 地雷要素" の場合、地雷可能性マップの対応する要素を、" 可能性あり" にします。
- 記憶用配列の要素が、" 空要素" の場合、空可能性マップの対応する要素を、" 可能性あり" にします。
- 記憶用配列の要素が、" 独立要素" の場合、地雷可能性マップの対応する要素と、空可能性マップの対応する要素を、両方とも" 可能性あり" にします (前述したように、独立要素は、増えたり減ったり位置が変わったりしませんので、これは最初の 1 回だけ行えば済みます。)

全ての" 最終的に矛盾の生じないような地雷配置" に関して処理が完了した後、次のような方針で出力を行っていきます。

- "地雷要素である可能性がない" 要素は、確実に地雷が埋められていない要素です。
- "空要素である可能性がない" 要素は、確実に地雷が埋められている要素です。

矛盾のない初期マップ (すなわち、矛盾のない地雷配置が最低 1 つ以上あるような初期マップ) については、" 地雷の可能性もなく、空の可能性もない" 要素はない、ということに注意して下さい。

もちろん、ビット演算等を活用すれば、結果保存用の配列は 1 つで済ますことが可能です。

5 あとがき

さーて、これが筆者の書いた解答となるプログラムです、ババーン、と、プログラムリストを載せたいところですが、諸事情(という名の、実はプログラムをまだ書いていないという事情)により、それはできない相談です[5]。

入学前からプログラミングに興味があった新入生の方であれば、記事の内容通り(か、独自に編み出したアルゴリズム)のプログラムを書くのはそう難しいことではないと思いますので、ぜひ挑戦してみてはいかがでしょうか。というか余裕、という新入生は、是非とも MMA の戦力となってください。

また、こんな記事読んでも、意味が分かんないよ、という新入生も多いと思います。初期の予定では、もう少し丁寧に、アルゴリズムのアの字も知らないような方にも分かってもらえるような記事を書く予定だったのですが、筆者には荷が重すぎたようです。幸い現在では、internet を通じて、アルゴリズムやプログラミングに関する情報が大量に入手できますので、どんどん情報を得て、どんどんプログラムを書けるようになって下さい。

では、最後まで読んでくださった新入生の方が、「素晴らしい記事だった、感動した!!」と思ってくれたりして、ICPC というイベントに興味を持ってくれたりして、MMA の仲間となってくれることを願って、締めくくりの言葉としたいと思います。

[5]百萬石配布までに、http://www.mma.gr.jp/~xmoe/works/minesweeper に、プログラムを載せる事が出来るかもしれませんので、興味のある方はそちらを御覧ください。

Windows Scripting Host

佐野 浩司 (sano@mma.club.uec.ac.jp)

2001年11月

概 要

国産3DCGソフトShadeでのスクリプト作成のために勉強し始めたものですが、けっこう別な使い方もできることがわかったのでまとめてみました。

1 はじめに

Windows Scripting Host(以下WSH) は、簡単にいえばWindowsをスクリプトで操作しようというものです。言語はJScript、VBScriptを使用します。ここでは、JScriptを使用します。

詳しくは、ここら辺を参考にしてください。

- http://www.microsoft.com/japan/developer/scripting/default.htm
- http://www.asia.microsoft.com/Japan/Developer/Scripting/default.asp

JScriptは、Microsoft版Javascriptのようなもので、オブジェクトを基本とするスクリプト言語でインタプリタにより実行されます。それ単独では独立したアプリケーションは作成できず、またファイル入出力機能もサポートされていません。ファイル入出力には、FileSystemObjectオブジェクトモデルを使用します。

実行するには、スクリプトをテキスト形式で保存し拡張子を.jsにし、ダブルクリックします。実行できない場合は上のサイトで適当に必要なエンジンを拾ってください。

2 なんかつくってみる

それでは、ファイルの入出力を使う簡単なスクリプトを作ってみます。以下のスクリプトは、Internet Explorerの「お気に入り」フォルダにあるインターネットショートカットファイルを片っ端から読んで、スクリプトのあるフォルダにそれらのショートカットをまとめた「Favorites.html」を作成します。

リスト 1: Favorites.wsh

```
1  //ファイルの読み書きモードの設定
2  var ForReading = 1, ForWriting = 2, ForAppending = 8;
3  var TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0;
4
5  //フォルダの階層の深さを示す
6  var dep = 0;
7
8  //ファイルを扱うためにFileSystemObjectを使用します
9  var fs = new ActiveXObject("Scripting.FileSystemObject");
10
11 //htmlファイルを作成し、同名のファイルがあれば上書きします
12 fs.CreateTextFile("Favorites.html",true);
13 var html = fs.GetFile("Favorites.html");
14
15 //ファイルハンドルみたいなもの
```

```
16 var stream = html.OpenAsTextStream(ForWriting,TristateUseDefault);
17
18 //folder オブジェクトを取得
19 var favf = fs.GetFolder("C:\\WINDOWS\\Favorites");
20
21 //改行記号付で 1 行書く
22 stream.WriteLine("<html><body>");
23
24 //いわゆる関数呼び出し
25 seekFavorites(favf);
26
27 //同じく 1 行書く
28 stream.WriteLine("</body></html>");
29
30 //ファイルハンドルを閉じる
31 stream.Close();
32
33 //関数定義
34 function seekFavorites(f){
35   var urlline,url,sitename,fitem,name,urlre,namere;
36   var subf = new Enumerator(f.SubFolders);
37     // f で与えられたフォルダオブジェクト中のサブフォルダを、
38   for(; !subf.atEnd(); subf.moveNext()){
39     // Enumerator オブジェクトでリストのようにして返している。
40     for(i=1; i <= dep; i++){
41       // そのサブフォルダごとに、seekFavorite を再帰呼び出ししている。
42       stream.Write("  |  ");
43     }
44     stream.WriteLine("<b>" + subf.item().Name + "</b><br>");
45     dep++;
46     seekFavorites(subf.item());
47     dep--;
48   }
49   var fav = new Enumerator(f.Files);
50     // Enumerator オブジェクトでフォルダ中のファイルのリストを返している。
51   for(; !fav.atEnd(); fav.moveNext()){
52     favfile = fav.item();
53     sitename = favfile.Name              // ファイルの名前を得る。
54     namere = /\.url/;
55     name = sitename.replace(namere,"");
56       // ファイル名は「サイト名.url」と言う形式なので、
57       // 正規表現を用い余計な「.url」を消している。
58     ff = favfile.OpenAsTextStream(ForReading,TristateUseDefault);
59     ff.SkipLine();
60       // ファイルを読み込み、1 行飛ばす。
```

```
61      urlline = ff.ReadLine();              // 1行読み出す。
62      urlre = /URL\=|BASEURL\=/;
63      url = urlline.replace(urlre,"");
64        // ファイルの2行目に、「URL=サイトURL」の形式でURLが
65        // 記述されているので、余計な「URL=」を消す。
66      for(i=1; i <= dep; i++){
67        stream.Write("  |  ");
68      }
69      stream.WriteLine("<a href=" + url + ">" + name + "</a><br>");
70    }
71  }
```

結果として、以下のようなHTMLファイルが作成されます。

リスト 2: Favorites.html(表示イメージ)

```
search engines
  |  LYCOS Japan
  |  Excite Japan サーチストリーム
  |  Google
  |  IP ドメイン SEARCH
  |  Google image search
  |  BrowserCrasherChecker
  |  NAVER Japan
Shade
  |  plugins
  |    |  site.k3.watanabe
  |    |  Shade 増強大辞典
  |    |  Cotton's Graphic Studio
  |    |  HK WORLD
  |    |  Shade スクリプト講座
  |    |  STONE   FIELD   MENU
```

リスト 3: Favorites.html(HTML ソース)

```
<b>search engines</b><br>
  |  <a href=http://www.lycos.co.jp/>LYCOS Japan</a><br>
  |  <a href=http://village.infoweb.ne.jp/~fwif8089/genkan/odekake/linkpage.htm>Excite Japan サー
チストリーム</a><br>
  |  <a href=http://www.google.com/intl/ja/>Google</a><br>
  |  <a href=http://www.mse.co.jp/ip_domain/>IP ドメイン SEARCH</a><br>
  |  <a href=http://images.google.com/>Google image search</a><br>
  |  <a href=http://www.jah.ne.jp/~fild/cgi-bin/LBCC/lbcc.cgi?>BrowserCrasherChecker</a><br>
  |  <a href=http://www.naver.co.jp/>NAVER Japan</a><br>
```

```
<b>Shade</b><br>
  |  <b>plugins</b><br>
  |    |  <a href=http://www.kei3.com/>site.k3.watanabe</a><br>
  |    |  <a href=http://village.infoweb.ne.jp/~fwhi6430/Shade/>Shade 増強大辞典</a><br>
  |    |  <a href=http://www.cottons.org/>Cotton's Graphic Studio</a><br>
  |    |  <a href=http://www1.odn.ne.jp/~cam77430/index.html>HK WORLD</a><br>
  |    |  <a href=http://www.kt.rim.or.jp/~mixjam/script_course/>Shade スクリプト講座</a><br>
  |    |  <a href=http://www.bekkoame.ne.jp/~sight2/menu.htm>STONE   FIELD   MENU</a><br>
```

WSH では、Internet Explorer を操作することができます。上のスクリプトを使って、お気に入りフォルダの中のサイトを片っ端から巡回するスクリプトを作成します

リスト 4: Scan1.wsh

```
1  var ForReading = 1, ForWriting = 2, ForAppending = 8;
2  var TristateUseDefault = -2, TristateTrue = -1, TristateFalse = 0;
3  var fs = new ActiveXObject("Scripting.FileSystemObject");
4  var favf = fs.GetFolder("C:\\WINDOWS\\Favorites");
5
6  //Internet Explorer オブジェクトを得る。
7  var IE = WScript.CreateObject("InternetExplorer.Application");
8
9  //IE を表示する。
10 IE.Visible = true;
11
12 PatFavorites(favf);
13
14 function PatFavorites(f){
15   var urlline,url,sitename,fitem,name,urlre,namere;
16   var subf = new Enumerator(f.SubFolders);
17   for(; !subf.atEnd(); subf.moveNext()){
18     PatFavorites(subf.item());  //再帰処理
19   }
20   var fav = new Enumerator(f.Files);
21   for(; !fav.atEnd(); fav.moveNext()){
22     favfile = fav.item();
23     ff = favfile.OpenAsTextStream(ForReading,TristateUseDefault);
24     ff.SkipLine();
25     urlline = ff.ReadLine();
26     urlre = /URL\=|BASEURL\=/;
27     url = urlline.replace(urlre,"");
28     IE.Navigate(url);   //IE で URL を表示
29     WaitLoad();
```

```
30    }
31  }
32
33  function WaitLoad(){          //ページのロード完了まで待つ
34    if(!IE.Busy && IE.Document.readyState == "complete"){
35      WScript.Sleep(3000);
36      if(!IE.Busy && IE.Document.readyState == "complete"){
37        return;
38      }
39      else{
40        WScript.Sleep(3000);  //3000ミリ秒スクリプトを停止
41        WaitLoad();
42      }
43    }
44    else{
45      WScript.Sleep(3000);
46      WaitLoad();
47    }
48  }
```

また、IE を使ってできることはただページを表示することだけではありません。IE.Document で IE 上の document オブジェクトを得ることができます。パスワードを入力しなければ進めないページにもフォーム入力を行うことにより、巡回することができます。

リスト 5: Scan2.wsh

```
1  var IE = WScript.CreateObject("InternetExplorer.Application");
2  IE.Visible = true;
3  IE.Navigate("http://www.host.com/cgi-bin/bbs.cgi");
4  WaitLoad();
5  IE.Document.forms[0].my_name.value = "guest";
6  IE.Document.forms[0].my_pass.value = "hogehoge";
7  IE.Document.forms[0].button.click();
8
9  function WaitLoad(){
10   :
11   :
12 }
```

これらのようにして得られたページの内容（ソース）は、IE.Document.body.innerHTML で得られるので、これはこれでまた別に利用できそうです。

3 最後に

これらのスクリプトはWSH上以外にもIE上で動いてしまうので、(一応警告は出る)変なスクリプトを実行してしまわないように普段はセキュリティーレベルを上げる(ていうか使わない)ようにしましょう。っていうかそれDUKEじゃん、みたいなスクリプトも作れてしまうので、悪用は止めましょう。

参考サイト

- http://member.nifty.ne.jp/aya/index.htm
- http://www.microsoft.com/japan/developer/scripting/default.htm
- http://www.asia.microsoft.com/Japan/Developer/Scripting/default.asp

PML mini HOWTO

菊池 孝治 (xmoe@mma.club.uec.ac.jp)

2002年4月

概要

この記事では、筆者がへなちょこ web 日記を記述するために開発した、PML (Plain Markup Langage) と、その処理系である pmlexec について、簡単な紹介をしたいと思います。pmlexec は、現在、任意の PC-UNIX 上で動作することを目標としての開発段階にあります。

1 はじめに

1.1 この記事は何であって何でないのか

この記事は、できたてほやほや、世界初公開の、PML の紹介記事です。筆者の文章力とページ数 (と時間) の関係で、PML について正確に全てを語りつくすのは不可能ですので、PML の仕様書、というわけではありません。

「正確で、より深いことを知りたい方は、参考文献を参照して下さい」と言いたいところですが、何しろ、できたてほやほやの言語ですので、参考文献すらありません。むしろ、「正確で、より深いことを知りたい方は、使ってみて、バグを出して、ソースを読んで、バグを直して、参考文献を書いて下さい。」とお願いしたいくらいです。

そんな訳で、この記事は、言ってみれば、

あなたのための、PML ブートローダーです。

しょうがねぇな、ロードしてやるか、という優しいあなたに求められるスペックは、

- BSD , linux 等の PC-UNIX が操作できる。(レポートを自分で書いて、提出できるくらい?)
- できれば、Perl が読める 。(というか、Perl 初心者である筆者の書いたプログラムを眺めて、何をやっているのかがだいたい分かる。というのは、PML を利用するのに Perl が書ける能力が必須というわけではないのですが、筆者が書いたサンプルは、Perl を利用していますので、Perl を読めたほうが、理解しやすいのでは、と思われるからです。)
- 標準入力から文字列を読み込み、加工して、標準入力に文字列を出力するようなプログラムが書ける。(C でも、Perl でも、Ruby でも、がんばればシェルスクリプトでも OK かもしれません。)
- できれば、HTML を自分で書いたことがある。(PML で書いた文書は、いまのところそのままブラウザに送ったりしても、ブラウザがそれを理解して奇麗に表示してくれはしませんので、HTML 等に変換することになります[1]。HTML でなくても良い (XML, プレインテキスト等) のですが、サンプルでは HTML を利用しているため、HTML を書いたことがあるほうが、理解しやすいと思われます。)
- できれば、LaTeX で文章を書いたことがある。(というのは、PML は LaTeX からアイデアを頂いているので、LaTeX で何かを書いたことがあれば、わかりやすいのではないかと思われるからです。といっても、LaTeX と似ているのは表面的で限られた部分だけですので、LaTeX が母語、といった方には逆に分かりにくいかもしれません。)

といったところだと思います。

ではでは、楽しい (かもしれない)、PML の世界のはじまりはじまり〜。

[1] そういう意味で、PML = Pre Markup Language かもしれません。

1.2 まずは材料です

とりあえず、pmlexec のソースファイル一式をダウンロードして、コンパイルしてみることから始めてみましょう。

pmlexec の公式 web ページ(というほどのものではありませんが)は、**http://www.mma.gr.jp/~xmoe/works/pmlexec/** です。ここから最新の tarball (pmlexec-hogehoge.tar.gz) をダウンロードして、お好きなディレクトリに展開して下さい。

この tarball は、GNU の autoconf/automake を利用して作成されていますので、

```
[xmoe@localhost pmlexec-2002-03-25] $ ./configure
    ...[省略]...
[xmoe@localhost pmlexec-2002-03-25] $ make
    ...[省略]...
```

としてみましょう。運がよければ、src ディレクトリの下に、pmlexec という実行ファイルが作成されるはずです。

運悪く、make の途中でエラーが起こってしまった貴方は、すぐさまパッチを書いて筆者に連絡をしましょう。世界的なソフトウェアの contributer になれるかもしれないチャンスです[2]。

1.3 雰囲気だけでも …

さて、無事に実行ファイルの作成が終ったところで、サンプルスクリプトを実行[3]してみましょう。とりあえず、samples/basic ディレクトリの sample1.pml というファイルを眺めてみます。

リスト 1: sample1.pml

```
1  #filter <html.pmlf>
2
3  \begin{html}{ja-JP}
4  \begin{htmlbody}
5
6  \h3{pmlexec サンプル (HTML)}
7
8  \hr
9
10 これは、pmlexec を利用して、html ファイルを作成する場合の
11 サンプルファイルです。\br
12
13 ...[省略]...
14
15 \h4{リンク}
16 link コマンドや url コマンドを使うと、
17 \begin{itemize}
18 \item
19   \link{http://www.mma.gr.jp/~xmoe}{他のページへのリンクを貼ることができます。}
20 \item
21   \link{mailto:xmoe@mma.gr.jp}{メールアドレスでも大丈夫です。}
```

[2]本当のことを白状しますと、現在、形だけは autoconf を利用していますが、肝心のソースファイルで環境依存性の吸収をしていない段階ですので、「make ができた/できなかった」という情報だけでも貴重です。是非とも情報提供をお願いします。

[3]PML を使って、/etc 以下のファイルを全て消去する、といった便利なことも、やろうと思って、必要な権限さえあればできてしまいますので、ソースファイルをコンパイルではなく、あえて「スクリプトを実行」という言葉を使っています。

```
22 \item
23   \url{http://www.mma.gr.jp}
24 \item
25   \url{mailto:xmoe@mma.gr.jp}
26 \end{itemize}
27 url コマンドでは、リンク先の URL がそのまま表示されます。
28
29 ...[省略]...
30
31 \h4{"記号"の展開}
32 "ダブルクオート" や &アンパサンド&、<大小記号>は、自動的に展開 (?) されます。\br
33 HTML のソースコードを見てみて下さい。
34
35 ...[省略]...
36
37 #include "footer.pml"
38
39 \end{htmlbody}
40 \end{html}
```

何やら LaTeX 風にマークアップされた文書ですね。これを" 実行" してみます。

```
[xmoe@localhost basic] $ ../../src/pmlexec -F ../pmlf -o sample1.html sample1.pml
```

これで、同じディレクトリに、sample1.html というファイルが作成されたと思います。sample1.html を w3m や netscape 等のブラウザで表示してみましょう。

どうでしょう。サンプル中の文章にあるように HTML でマークアップされていると思います。

これじゃ単なる置換プログラムじゃん、こんなの 簡単な Perl プログラム でもできるよ[4]、と思ってしまった方は、他のサンプルファイルを覗いて見て下さい。もう少し抽象度の高いコマンド (footnote や、refbook 等) が利用できるのがお分りになると思います。

2 pmlexec は何をしてくれるの?

2.1 大雑把な話

前章の最後の節でみたように、ぶっちゃけた話、

PML とは
: LaTeX みたいなマークアップ言語

pmlexec とは
: PML でマークアップされた文書を、何か他のフォーマット (HTML 等) に変換するソフト。

ということにしておけば、大きな間違いではないと思います。この章の次節以降では、リスト 1 を参考にしながら、PML と pmlexec について、より詳しく見ていくことにします。

[4] PML = Poor Markup Laanguage という話もありますが。

2.2 filter ディレクティブ

まずは1行目の、

```
#filter <html.pmlf>
```

から始めましょう。

C言語などをご存じの方はご想像がつくと思いますが、#で始まる行は、pmlexec に対するディレクティブ[5]です。filter ディレクティブでは、「フィルタ」を指定します。では、その「フィルタ」とは、一体何物なんでしょうか、ということで、配布物の、samples/pmlf ディレクトリ[6]中にある、html.pmlf というファイルをみてみましょう。

リスト 2: html.pmlf

```
1  #! /usr/bin/perl -w
2
3  require 'pmlf.pl';
4
5  ...[省略]...
6
7  # templates
8  my %template;
9  $template{'strong'}          = '1:<STRONG>arg1</STRONG>';
10
11 ...[以下省略]...
```

ファイルの実行フラグも立っていますし、リスト2の1行目の記述からしても、これが、Perl で書かれた実行可能ファイルであることがお分かりいただけると思います。

そうです。「フィルタ」というのは、実行可能ファイルのことだったのです。sample1.pml というスクリプトが実行されると、その中でフィルタとして指定された、html.pmlf というフィルタも、実行されます。

また、この例では、フィルタが Perl で書かれていますが、Ruby や C 言語 でも実行可能ファイルを作成することが可能ですので、Ruby や C 言語 で PML 用のフィルタ (PML filter) を書くことも可能です。

一つのスクリプト中で、複数のフィルタを指定することも可能だったり (他のサンプルを覗いてみて下さい) しますが、そういった込み入った話や、フィルタは何をする実行可能ファイルなのか、という話は後述しますので、今は以下に示すことを覚えておいて下さい。

> pmlexec は、PML で書かれたスクリプトを構文解析し、ディレクティブや、次に説明するコマンドを認識して、実際のコマンド、文字列の変換処理は、スクリプト上で指定されたフィルタ (実行可能ファイル) を実行し、それにお任せ (委譲) する。

2.3 コマンド

さて、リスト1の3行目から40行目に渡って出現する、

[5] ディレクティブ：指示子とも呼ばれるもので、プログラムなどで、通常の行よりも高いレベルの、「言語処理系に対する指示」などをするために使われます。

[6] リスト1では、フィルタの名前が、() で囲まれていますね。これは、「コマンドラインの -F (–filter-directory) オプション で指定されたディレクトリから指定のファイルを探せ」といった意味になります。ファイル名を"" で囲んだ場合は、実行時のワーキングディレクトリから指定のファイルが探されます。

```
\link{http://www.mma.gr.jp/~xmoe}{他のページへのリンクを貼ることができます。}
```

といった、バックスラッシュと、それにつづく半角英字の文字列[7]、それにつづく、{} や [] で囲まれた 0 個以上の引数の部分が、" コマンド" です。

コマンドは、いってみれば PML の主役、なわけで、フィルタの主な役割は、これらのコマンドを、他の文字列 ("テキスト" と呼びます) に変換することにあります。

2.4 include ディレクティブ

最後に、リスト 1 の 37 行目にある、

```
#include "footer.pml"
```

という行です。この行は、#filter と同様、ディレクティブです。

C 言語などでご存じの方には説明の必要がないと思いますが、" " で囲まれたファイル名のファイルをそこに「含め」ます。この例では、sample1.pml と同じディレクトリ[8]にある、footer.pml というファイルの内容が、そこに挿入されます。

3 PML filter を書いてみよう

3.1 まだ良く分かってないのに、いきなりフィルタを書けるの?

PML と pmlexec について、大体のところはつかめたと思いますので、簡単なフィルタを書いてみることにします。もうちょっと詳しいことを知ってから、と思われる方もいらっしゃると思いますが、習うより慣れろ、という方向で。。。

3.2 単独で動作するフィルタの例

一番目の例として、

now コマンドがあると、そこに実行時の時間を挿入してくれる。

という単純なフィルタを Perl で書いてみることにしましょう。

まずは、サンプルとなるスクリプトです。配布物の中の、samples/basic ディレクトリに、下記のようなファイルを、文字コードは EUC で作成してみて下さい。

リスト 3: practice1.pml

```
1 #filter <now.pmlf>
2
3 現在の時刻は、\now です。
```

さて、次にフィルタを書いていきますが、基礎知識として、次のことを覚えて下さい。

- 各フィルタは、標準入力と標準出力を使って、pmlexec と通信します。

[7]細かいことをいうと、先頭文字が半角英字かアンダーバー、それにつづく文字が半角英数字かアンダーバーであり、最初の文字ではない最後の文字はアスタリスクであっても良い、一文字以上の文字列です。

[8]filter ディレクティブと同様、ファイル名が () で囲まれている場合は、コマンドラインの -I (–include-directory) オプションで指定されたディレクトリにあるファイルが挿入されます。

- コマンドや通常の文字列は、コマンドなのか通常の文字列なのか区別できる形で、フィルタに送られてきます。
- フィルタの仕事は、それらを加工して出力するか、もしくは何もしないで出力するか、またはエラーと判断して終了するか、です。
- フィルタを書くためには、pmlexec との通信プロトコルに従って、入力と出力をしなければいけませんが、Perl でフィルタを書く場合、そのプロトコルをカプセル化した、"pmlf.pl" というパッケージが利用できます。

ということで、配布物の samples/pmlf ディレクトリに、下記のようなファイルを作って (文字コードは EUC で)、実行可能にしてみましょう。

リスト 4: now.pmlf

```
1  #!/usr/bin/perl
2  require 'pmlf.pl';
3
4  while (&pmlf::get_pml_command (\$command ,\@arguments)) {
5      if ($command eq 'now') {
6          ($sec, $min, $hour, $mday, $mon, $year, $wday, $yday, $isdst) = localtime(time);
7          $year += 1900;
8          $mon += 1;
9          &pmlf::put_pml_text ("$year 年$mon 月$mday 日$hour 時$min 分$sec 秒");
10     }
11 }
12
13 exit 0;
```

では、リスト 4 について多少の解説を。

まず、1 行目の

```
#!/usr/bin/perl
```

については、解説の必要もないかもしれませんが、「このファイルは実行可能なスクリプトであって、インタプリタとして /usr/bin/perl という実行可能ファイルを使ってね。」ということを OS のカーネルに教えてあげるための行です。/usr/bin/perl という指定については、実行する環境ごとに、適切なものにして下さい。

2 行目の、

```
require 'pmlf.pl';
```

という行は、「"pmlf.pl" というパッケージを使いますよ。」ということを、Perl インタプリタに教えてあげるための行です。前述したように、samples/pmlf ディレクトリにある "pmlf.pl" というパッケージには、pmlexec と フィルタの間の通信プロトコルがカプセル化されています。

さて、4 行目から 11 行目にかけての、while ループが、このフィルタの、いってみれば「メイン」となります。

4 行目の、

```
&pmlf::get_pml_command (\$command ,\@arguments)
```

という関数呼び出しが、pmlexec から、コマンドを入力している部分です。

その後、5 行目の、

```
if ($command eq 'now') {
```

という部分で、"now" というコマンドが入力されたかどうかを判定し、入力されたならば、6,7,8 行目で現在時刻を求め、9 行目の、

```
&pmlf::put_pml_text ("$year 年$mon 月$mday 日$hour 時$min 分$sec 秒");
```

という部分で、pmlexec に対して、結果となる「テキスト」を出力しています。

このプログラムのポイントを一つあげるならば、

入力されたコマンドが now コマンドでなければ、何もせずに、次のコマンドを得ている。

ということです。これは、pmlexec との通信のポイントではなく、pmlf.pl というパッケージの使いかたに関する部分なのですが、pmlf.pl では、get_pml_command が呼び出された後、put_pml_text 、または、後述する put_pml_command が一度も呼び出されずに、再度 get_pml_command が呼び出された場合、一度目の get_pml_command で入力されたコマンドを加工せずに素通しします。

逆に言えば、get_pml_command であるコマンドが入力され、それに対して何も出力したくない場合は、

```
&pmlf::put_pml_text ("");
```

と、明示的に指定してやる必要があるということです。

解説が一通り済んだので、実行してみます。pmlexec は、出力ファイルの指定 (-o, -output-file オプション) がない場合、実行結果を標準出力に出力します。

リスト 5: 実行結果

```
[xmoe@localhost basic]$ ../../src/pmlexec -F ../pmlf practice1.pml
現在の時刻は、2002 年 3 月 31 日 5 時 43 分 23 秒 です。
```

見事に意図した通りの出力を得ることができました。

3.3 他のフィルタに依存するフィルタの例

一つのスクリプトの中で、複数のフィルタを指定できる、ということは前述しました。

ここでは、そのように使われるフィルタの例として、ヤフーオークションの、" オークション ID" を引数として渡すと、そのオークションのページにリンクをしてくれるようなフィルタ、"yauction.pmlf" を作成してみたいと思います。

"リンクをする" というのをもう少し具体的に定義してみると、

PML スクリプトを用いて、HTML で記述された文書を作成し、その中で、yauction コマンドで指定されたオークション ID を持つオークションのページを対象とする、アンカータグを自動的に記述する。

といったことになると思います。

PML スクリプトを用いて、HTML を作成することになりますが、リスト 1 で利用されている"html.pmlf" というフィルタが存在するわけですので、html に関するフィルタを一から書く必要はありません。

早速、下記のような内容のファイルを、samples/basic ディレクトリ中に、EUC で作成してみましょう。

リスト 6: practice2.pml

```
1  #filter <html.pmlf>
2  #filter <yauction.pmlf>
3  #filter <now.pmlf>
4
5  \begin{html}{ja-JP}
6  \begin{htmlbody}
7
8  \h4{物欲りすと (\now 作成)}
9  \begin{itemize}
10  \item \yauction{e9444179}{500 円っていうのは安いかも。}
11  \item \yauction{43783509}{同じ通販ものとはいえ、こっちは人気商品なんだよねぇ。}
12 \end{itemize}
13
14 愛機の ThinkPad560 も逝ってしまったし、
15 \yauction{44631443} とか \yauction{43758840} とか買えたらいいなぁ。
16
17 \end{htmlbody}
18 \end{html}
```

一つ注意しておきたいのは、filter ディレクティブの順序です。

PML スクリプトには、「他のフィルタから依存されているフィルタ」を先に、「他のフィルタに依存しているフィルタ」を後に書く必要があります。リスト 6 の例で言えば、yauction.pmlf というフィルタは html.pmlf というフィルタに依存する予定ですので、html.pmlf を先に、yauction.pmlf を後に書いておきます。

次に、yauction コマンドを処理するための、yauction.pmlf を書いてみます。

samples/pmlf ディレクトリに、以下のようなファイルを作成し、実行可能にしてみましょう。

リスト 7: yauction.pmlf

```
1  #!/usr/bin/perl
2  require 'pmlf.pl';
3
4  sub yauction_uri {
5      # オークションページの URI を作成
6      $id = $_[0];
7      $uri = "http://page";
8      if ($id =~ /^[a-z]/) {
9          $uri = $uri . (ord ($id) - ord ('a') + 1);
10     }
11     $uri = $uri . ".auctions.yahoo.co.jp/jp/auction/$id";
12     return $uri;
13 }
```

```
14
15 while (&pmlf::get_pml_command (\$command ,\@arguments)) {
16     if ($command eq 'yauction') {
17         if ($#arguments==0) {        # 引数の数が 1 つの場合
18             &pmlf::put_pml_command ('link', &yauction_uri ($arguments[0]),
19                                     "オークション $arguments[0]");
20
21         } elsif ($#arguments==1) {  # 引数の数が 2 つの場合
22             &pmlf::put_pml_command ('link', &yauction_uri ($arguments[0]),
23                                     $arguments[1]);
24
25         } else {                     # その他の場合はエラー
26             print STDERR 'illegal yauction command';
27             exit 1;
28         }
29     }
30 }
31
32 exit 0;
```

18 行目、22 行目に、

```
&pmlf::put_pml_command ('link', &yauction_uri ($arguments[0]), $arguments[1]);
```

といった記述があります。

PML スクリプト内で link コマンドを記述すると、html.pmlf というフィルタが、それを HTML のアンカータグに置き換えてくれるように、html.pmlf よりも前段にあるフィルタで、put_pml_command を呼び出して、link コマンドを発行することにより、html.pmlf が、それを HTML のアンカータグに置き換えてくれます。

ということで、リスト 6 のスクリプトを実行するには、samples/basic ディレクトリ以下で、下記のように (シェルに対して) コマンドを入力します。

```
[xmoe@localhost basic] $ ../../src/pmlexec -F ../pmlf -o practice2.html practice2.pml
```

実行後、w3m 等のブラウザで、practice2.html を表示の表示を確認してみましょう。

4 pmlexec はどうやって動いてるの?

4.1 ソースを読もう!!(というか読んであげて下さい)

ということで、この章では、pmlexec の内部的なことについて、ポイントと思われることのみですが、記述していきたいと思います。

正直、行き当たりばったりでコードを書いてしまった部分が多いため、お世辞にも素晴らしいコードとはいえないと思いますので (hoge_hoge と pagePage という書き方が混在してたり)、この章の内容が、pmlexec をいじってやろう、とか、僕もなにか言語処理系を書いてみようかなぁ、という方のお役に立てば、と思います。

4.2 フィルタの実行

まずは、フィルタの実行の仕組みです。(ソースファイルでいうと、evalution.c がこれにあたります。)

単一のフィルタを実行する場合、話は単純で、子プロセスとの通信用のパイプを用意し、fork (3) した後、子プロセスのほうで、標準入力と標準出力を、用意したパイプと入れ換えてしまい、/bin/sh を通して、filter ディレクティブで指定されたコマンドを実行しています。

また、薄々勘づいている方もいらっしゃると思いますが、複数のフィルタを実行する場合も、pmlexec 内部の処理は至極簡単でして、/bin/sh にコマンドを渡す際に、「パイプでつないでね」という指示を出して、複数のコマンドを実行しているだけです。

つまり、

```
#filter <html.pmlf>
#filter <footnote.pmlf>
```

というディレクティブによって、フィルタが指定された場合、内部的には、

```
/bin/sh -c "(footnote.pmlf) | (html.pmlf)"
```

というようなコマンドが実行されています。

4.3 字句解析とディレクティブ処理

字句解析は Flex を利用しています。(ソースファイルは、scanner.l です。)

ディレクティブの処理は、この段階でしてしまっていますので、

```
\begin
#include "/dev/null"
{itemize}
\end{itemize}
```

というスクリプトがエラーになることはないと思います。

また、バックスラッシュによるエスケープの処理もこの段階で行い、構文解析の単純化を計っています。

4.4 構文解析とコマンド/テキスト処理

構文解析は Bison を利用しています。(ソースファイルは、paser.y です。)

ひたすら、文字列をリストでつないでいくような処理なのですが、一つだけポイントを挙げるとすると、「要素となる文字列を逆順につないでいる」ことでしょうか。

利用しているリストは、シンプルな単方向リストですので、素直に

```
connectList (pList, pListA);
connectList (pList, pListB);
connectList (pList, pListC);
```

と書いてしまうと、pList 中の要素が増えるにしたがって、処理に時間がかかるようになってしまいます。(といっても、他の処理にかかる時間に比べると、無視できるほどかもしれませんが。。。)

そこで、pmlexec では、

```
pList = connectList (pListA, pList);
pList = connectList (pListB, pList);
pList = connectList (pListC, pList);
pList = reverseList (pList);
```

という形で、構文解析中のリスト処理を、要素の数に比例する計算量で処理できるようにしています。

もしかすると、これは、Bison や Yacc を利用する際の、お婆ちゃんの知恵袋的なテクニックかもしれません。(激しく既出のような気もしますが。)

5 未来へむけて

5.1 TODO List

さて、なにしろできたてほやほやなので (くどい)、改良したいことは、細かいものから大きなものまで、たくさんあります。細かいものは、配布物の doc/TODO ファイルを参照していただきたいのですが、大きなものとしては、

バグを減らす

どんなプログラムを書くにしても、悩みの種なのがこれです[9]。現在、引数の中に巨大な文字列を入れるとコアダンプする、というバグがあることが確認されています。

エラーメッセージを分かりやすくする

現在、入力中に間違いがあると、片言の英語 (のような言語) でエラーメッセージが出力されますが、これをもう少し" インテリジェント" なものにしたいと考えています。文法エラーの時に、"parse error" というメッセージだけではあまりにも不親切ではないかとか、未定義のコマンドがあったときに、コマンド名しか出力しない (たとえば、begin{htlm} というコマンドが未定義であっても、"begin" は未定義です、としか言わない) のはいかがなものか、とかいう考えがあります。

フィルタの依存性に関するチェック機構

これは、筆者が最重要だと思っている項目です。例えば、配布物のサンプル中に、"footnote.pmlf" というフィルタがありますが、このフィルタは、後段に"link" や"br" 等のコマンドを処理してくれるフィルタが存在することを前提として処理を行っています。配布物中では、"html.pmlf" というフィルタがそれにあたるわけなのですが、その情報はどこにも書いていないし、エラーになってしまったところで、ユーザとしては、何かが足りないことは分かっても、何が足りないのかは分からないこともあり得るわけです。これに関する解決策として、

1. footnote.pmlf は、起動時 (もしくは最初の footnote コマンド受け付け時) に、"\require_pmlf{html.pmlf}" にあたるコマンドを発行する。
2. html.pmlf は、"\require_pmlf{html.pmlf}" というコマンドが入力された場合、それを消化 (後段にそれを素通ししない) する。
3. pmlexec は、require_pmlf コマンドが未消化のまま入力された場合、エラーとする。

というアイデアがあるのですが、現在、実装されていません。

フィルタからのエラー、警告などの統一的処理

現在の pmlexec の実装では、フィルタの処理中にエラーが発生した場合やおかしな入力があった場合に、フィルタが勝手に exit したり、標準エラー出力に警告メッセージを出力するだけです。これでは、pmlexec でエラーをコントロール (コマンドラインオプションによっては、エラーを無視したり、警告で停止したり) することができません。

前述の依存性チェックと同様に、*error コマンドや *warning コマンドをフィルタが発行することによって解決できるかもしれませんが、現在、未実装です。

[9] 「どんな」という言葉は、自分の中では NG ワード (その言葉が出てきたら、よく考えてみましょうという言葉) なのですが、この場合は「どんな」といっても言い過ぎではないように思えます。バグがあっても構わないプログラム (演習のレポート用のプログラムとか、プログラムのバグに関する研究の対象となるプログラム) は例外かもしれませんが。

プロセス数を減少させる

一つのコマンドの処理を考えると、pmlexec → フィルタ A → フィルタ B → pmlexec といった形で、シーケンシャルに処理が進んでいきますので、フィルタとして複数のプロセスが同時に動く必要性はありません。OS にとってオーバーヘッドが大きい、プロセスの生成に頼りすぎ (いわゆる富豪的) な気がします。

国際化

これは筆者がかなり不勉強な部分です。「EUC が通ればとりあえず良いでしょ。」というのが筆者の現状での態度なのですが、無知のために何か重要なことに気付いていないという可能性が高いので、詳しい方がいればぜひぜひ教えて下さい。

というところでしょうか。

筆者一人の知識と時間では、3 千光年くらいかかってしまう[10]と思われますので、興味がある方はどんどんいじってあげて下さい。

こういう機能も必要だ!!とかいう提案も大歓迎[11]ですので、なにかしら改良してしまった方は、ドシドシ筆者まで送ってください。

5.2 さらなる可能性を求めて

「さらなる可能性」というと大袈裟ですが、PML のシンプルで拡張が簡単な点を考えると、

internet 上で隆盛している web 掲示板システムへの応用

というのは、かなり現実味のある話だと思います。

現在の web 掲示板システムは、いわゆるスレッド式で最新の話題がページの上部に表示されるようなものや、ツリー方式で各記事のタイトルと親子関係を一覧できるものなど、システム上ではパソコン通信時代からの進歩がみられますが、肝心の投稿内容としては、テキストと低レベルな HTML タグ[12]が中心で、web の特性を充分に生かしていないというのが現状です。また、HTML タグを直接書き込めるようにしてしまうと、悪戯目的の投稿に対する対策が難しい、という問題もあります。

PML で投稿ができるようにすることで、HTML タグを直接書き込めるようにする必要もありません。また、掲示板のカテゴリによっては、そのカテゴリ限定の、新たなコマンドを定義することも可能です。例えば、think pad 関連の掲示板で、refpc コマンドによって、ある機種に関するページを参照できたりしたら面白そうだったりしませんか? ひいては、その掲示板の内容の密度の高さにも寄与できるかもしれません。

CGI への応用を考えた場合、現在の pmlexec の実装の不安定さや、プロセスを富豪のように生成する点は問題かもしれませんが … 。

6 あとがきにかえて

本を一冊読み終わった読者は、最低でも著者からなにがしかの褒め言葉をもらう権利がある。

という言葉がありますが、この記事は一冊の本ではないので、残念ながら、ここには褒め言葉はありません。あとがきから読みはじめるという習慣がある方は、いますぐ概要から読み直して下さい (笑)[13]

かわりに、というとなんですが、一つミミヨリかもしれな情報を。

---

[10] いや、比喩的に表すと、そのぐらい道のりがありそう、という意味ですので、「光年」っていうのは距離の単位だ!!というツッコミはなしという方向で … 。

[11] --no-thumbnails というオプションをつけた YO!!、とかいうパッチがきても、無視を決め込んだりはしません。

[12] ここで「低レベル」というのは、Perl からみた アセンブリ言語の抽象性の低さ、といったような意味です。HTML タグに恨みはありませんので念のため。

[13] というお約束のネタを一回書いてみたかったのです。

言語処理系をつくるというと、なにかしら難しいようなイメージがあるけど、Lex & Yacc (Flex & Bison) というソフトを使うと、結構簡単にできてしまうよ。

もちろん、筆者が言語処理系について云々するのを Java の設計チームが聞いたら、おそらくカチンとくるだろうと思われるように、「言語処理系をつくる」という言葉は、かなりの帯域を含んだものなのですが、この記事で話題にした PML のようなオモチャ言語の処理系であれば、学生プログラマが数週間 (桁外れの能力があれば、数日、もしくは数時間、さすがに数分間は無理だと思う) で作るのは難しいことではないと思います。

いくつかのコンピュータ言語を触ったことがある方ならば、自分でイチから言語を設計することによって、今まで見えなかったものが見えてくるかもしれません。実際、筆者もこの言語を設計するという作業を通じて、既知のコンピュータ言語の細かい、今まで気付かなかった点に気付くことができました。

では、ぜひ、PML で遊んであげて下さいね!!

参考文献

[1] John R.Levine,Tony Mason,Doug Brown 共著 (村上列 訳) : lex & yacc プログラミング , アスキー

分散システムのための Styx アーキテクチャ

Rob Pike
Dennis M. Ritchie
Computing Science Research Center
Lucent Technologies, Bell Labs
Murray Hill, New Jersey
USA
訳:山梨 毅 (uncover@mma.club.uec.ac.jp)
2001年11月

NOTE: Originally appeared in Bell Labs Technical Journal, Vol. 4, No. 2, April-June 1999, pp. 146-152.

1 概要

分散システムは、お互いに独立した各部分が集まって構成され、その集合体が場所的にも機能的にも分散していながら、統一された形を作りあげています。その例としてはネットワークＯＳ、インターネットサービス、国際電話通信網、さらに一般的には、今日のデジタルネットワークを使うすべての技術が分散システムといえます。しかしながら分散システムの設計、構築、保守は依然として困難なままであり、その主な原因は、簡単明瞭な相互接続のモデルが欠けているためだと思われます。

Plan 9 と Inferno という二つの分散システムを通して我々が得た経験は、上記のようなモデルを提案する助けになりました。これらのシステムは、ある有用なアイディアのもとして作られ、それが実際に有効であることを示していますし、またシステムの大部分において、そのアイディアができるかぎり利用されています。そのアイディアとは、システムの持つ資源を、階層的な名前空間の中に置かれたファイルとして提供する、ということです。ファイルとしてあらわされるものの実体は保存されたデータかもしれませんし、あるいはデバイス、動的情報のソース、サービスへのインタフェース、制御の参照口かもしれません。この手法によって、全てのシステム資源に対して統一された基本的な名前付け、構造付け、アクセス制御のメカニズムを提供することができるようになります。これを支える Styx という単純なネットワークプロトコルがこのアーキテクチャの中心となって、システム内通信のための共通の言葉を提供しているのです。

分散されてないシステムでさえ、サービスをファイルとして提供することで、使いなれた名前付け、アクセス制御、システム資源との結びつきを大幅に拡張できます。しかし、より重要なのは、リモートファイルシステムと接続するという良く知られた技術を使うことで、この手法が分散システムを構築するための自然な方法をとなるという事です。資源がファイルとして提供されていて、しかもリモートファイルシステムがあるのなら、それはすでに分散システムなわけです。ある場所で利用できる資源は、別の場所からも利用できるわけです。

2 導入

Styx プロトコルは 9P と呼ばれるプロトコルの一種です。9P は Plan 9 オペレーティングシステム [3] のために開発されました。簡単のために、ここでは Styx という名前を使うことにしますが、この二つは接続開始時の部分が違うだけです。

Styx の背景にあるもともとのアイディアは、クライアントとファイルシステムの間にあるファイル操作を、コンピュータネットワークで転送するためのメッセージに符号化するというものでした。この技術によって、Plan 9 はファイルサーバ-長期保存のファイルを集約的に貯蔵する場所- を、 CPU サーバ-大規模な共有メモリのマルチプロセッサ- とユーザ

の利用する端末から分離しています。このように、役割が異なるものを物理的に分離するという考え方がこのシステムの基本設計の中心にありました。従来ファイルシステムに関係ないと考えられていた様々な問題を解決するのに、このモデルを用いることができるというのは、我々も予期していませんでした。

ネットワークを介して計算資源を利用可能にするのに生じる困難の多くが、資源をファイルという形で提供することで、自然に解決されるということに気付いたのが大きな一歩となりました。これは、Styx がファイルという資源を透過的に外部に提供することができたからです。例えば、Plan9 のウィンドウシステムである 81/2 は、ローカルなハードウェアへのアクセスに対し、 /dev/mouse や /dev/screen といったファイルを提供する動的なファイルサーバとして実装されています。例えば /dev/mouse は、UNIXTM のデバイスファイルと同じように、通常のファイルとしてオープン、読み込みが行なえますが、 81/2 ではこのファイルが多重化されます。それぞれのクライアントプログラムは自分用の /dev/mouse を持っており、自分のウィンドウがディスプレイの中でアクティブな時にのみマウスのイベントが返されます。このような設計によって、マウスへのアクセスを制御する仕組が簡潔でわかりやすいものになります。しかしながら、その本当の長所は、ウィンドウシステムの資源をファイルで表現することで、Styx がそれらのファイル (= 資源) をネットワークを介しても提供できるようになることなのです。例えば、CPU サーバ上でグラフィックを利用したインタラクティブなプログラムを実行したいときには、単純に 81/2 が適当なファイルをそのサーバに提供すれば良いわけです。 Styx によって公開される資源はファイル-資源にはファイル名、ファイルパーミッション、ファイルアクセスの方法が与えられます—そのように振舞いますが、それらがディスク上に実際のファイルとして存在する必要はないことに注意してください。 /dev/mouse ファイルは通常の I/O 機能を介してアクセスされますが、これは実際には一時的なもので、実行中のプログラムによって動的に作りあげられているものなのです。それ自体には恒久的な実体は存在しません。

この手法をシステム全体に渡って進めていくことで、Plan 9 は資源開放の透過性について、目をみはるようなレベルを実現しました [5]。インタラクティブなグラフィックの他にも、デバッグや保守、ファイルバックアップ、果ては下層のネットワークハードウェアへのアクセスですら、Styx によってネットワークを介して利用することができます。そして、単なるファイルの I/O 以上に高度な技術を使うことなく、分散アプリケーションや分散サービスを構築することができるのです。

3 Styx プロトコル

Styx が占める位置というのは、Sun の NFS [2][1] や、Microsoft の CIFS[8] に類するものでしょう。しかしながら、Styx はこれらのものよりも単純で簡単に実装できます。さらに、NFS や CIFS はディスク上にある通常のファイルを共有するために設計されています。中でも NFS は、その下にある UNIX ファイルシステムの実装とキャッシュ方法と密接に結びついています。また、NFS や CIFS は Styx と異なり、 /dev/mouse といったような、動的に変化するデバイス的なファイルを外部に提供するのに向いていません。

Styx は階層的な木構造をしたファイルシステムの名前空間 [7] に加え、ファイルに関するアクセス情報 (パーミッション、サイズ、日付) や、ファイルを読み書きするための手段を提供します [7]。ユーザ (つまりアプリケーションプログラムを書く人たち) がそのプロトコルを直接知る必要はなく、かわりに彼らはファイルを参照して、それらの読み書きを行います。これで情報を提供したり変更したりしたことになるのです。

実際の運用では、他と通信を確立する、一台のマシン上で動作している Styx のクライアントと、その通信相手であるサーバが同じマシンで動作していても、また違うマシンで動作していてもかまいません。クライアント側の機能は、Plan 9 や Inferno のようにオペレーティングシステムに作り込まれていてもいいですし、またアプリケーションのライブラリでもかまいません。サーバもまた、オペレーティングシステムの一部であってもいいですし、まったく同様に、他のマシンで動作するアプリケーションであってもいいのです。どちらにしろ、クライアントとサーバはメッセージを交換することで通信します。そしてその結果、サーバに存在する階層的なファイルシステムが、クライアントから参照できる

というわけです。Styx プロトコルは、この交換されるメッセージに関する規格なのです。

ある階層では、Styx は以下のように分類される 13 種類のメッセージによって構成されています。

- 通信を開始する (ファイルシステムに接続する。)
- ファイルシステム内を探索する (つまり、必要なファイルの名前を指定し、それに対するハンドルを得る。)
- ファイルに対する読み書きを行う。
- ファイルの情報に対する照会、変更を処理する。

しかしながら、アプリケーションのプログラマはファイルに対するオープン、読み込み、書き込みの操作を記述するだけです。ライブラリあるいはオペレーティングシステムがこれらの要求を必要なバイト列に変換し、通信チャネルを通じて転送します。Styx プロトコルはこのバイト列の解釈をきちんと定義しています。このプロトコルは、ISO の OSI で言えば、だいたいセッション層に相当するものです。その仕様はマシンアーキテクチャの詳細とはほとんど無関係であり、実際、様々な命令セットやメモリレイアウトのマシンの間でうまく動作しています。プロトコルの概要を表 1 に示します。

名前	説明
attach	接続するユーザを認証する; FID を返す
clone	FID をコピーする
walk	FID を一つしたの階層に移動する
open	ファイル I/O のパーミッションをチェックする
create	新しいファイルを作成する
read	ファイルの中身を読む
write	ファイルの中身を書く
close	FID を開放する
remove	ファイルを削除する
stat	ファイル情報を得る: パーミッションなど
wstat	ファイル情報を変更する
error	失敗した操作のエラー状態を返す
flush	未実行の I/O 要求を無視する

表 1: Styx メッセージの概要

実際は、次のような操作は複数の Styx メッセージ列へと、システムの下層部分によって変換されます。

```
open("/usr/rob/.profile", O\_READ);
```

ファイルサーバへの接続が初めて確立されると、attach メッセージによってユーザ(ファイルをアクセスしようとしている人、あるいはエージェントプログラム)が認証され、 FID (file ID) と呼ばれるオブジェクトが返されます。この FID はそのサーバの階層のルートを表わしています。 open() が実行されると、次のように処理が進みます。

- ルートへの接続を保ったまま階層内を移動できるように、 clone メッセージによってルートの FID がコピーされ、新しい FID が返されます。
- 次に、 walk メッセージが複数発行され、この新しい FID がパスの中身を一つづつ移動 (usr, rob, .profile) してゆき、 /usr/rob/.profile ファイルへと辿りつきます。
- 最後に、このファイルを読み込めるかどうか open メッセージによってユーザのパーミッションがチェックされ、続く read あるいは write メッセージによる操作をこの FID に対して行うことが許可されます。

- I/O 操作が終了したら、 close メッセージによって FID は開放されることになるでしょう。

Styx の実装が一つ下の層に対して要求するのは、バイトストリームでの信頼できるトランスポート通信層だけです。例えば、Styx はインターネットプロトコル標準の転送制御プロトコルである TCP/IP か、 IP パケットを利用した、信頼できる順序ありデータグラムプロトコルであるインターネットリンク (IL) プロトコルの、どちらの上でも動作します。しかしながら、モデル自体は、構成要素を結ぶための実在するネットワークを必要としているわけではないことに注意してください。Styx は Unix のパイプや共有メモリ上でも十分動作します。この手法の強みは、ネットワーク上で動作するということではなく、ネットワークを通した場合でも、ローカルに実行される場合でも、そのどちらでも同じ動作をするということにあるのです。

4 アーキテクチャの手法

Styx はファイルシステムプロトコルの一つですから、資源をファイルとして提供するという、さらに大きなシステム設計手法を構成する要素でしかありません。この手法について議論するために、これからいくつかの例を挙げていきたいと思います。

ネットワーク通信の例

この例では、TCP/IP ネットワークへのアクセスが、Inferno や Plan 9 システムの中で、ファイルシステムの一部として、図 1 のような構成で表現されていることを示しています (一部省略されています。) [6]。

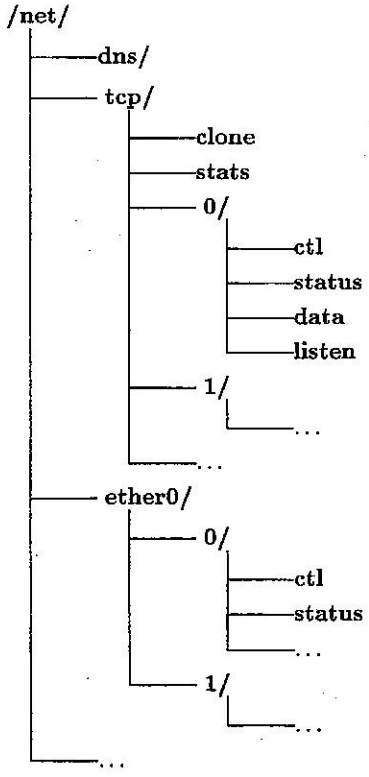


図 1: ファイルシステム上のネットワークの表現例

これは、/net/dns、/net/tcp/clone、/net/tcp/0/ctl といった「ファイル名」によって、ファイルを指定したり読み書きが行えるファイルシステムを表わしています。/net/tcp や /net/ether0 といったディレクトリもみられます。実際のネットワークインタフェースを持つマシンの場合であれば、これらファイルのように見えるものはすべて、TCP/ IP スタックを処理するカーネルのドライバによって構成されます。つまり、これらのファイルは実際にディスク上にあるファイルではないのです。これらの「ファイル」に対する操作はデバイスドライバへの操作へと変換されます。

あるアプリケーションが TCP/IP によって www.bell-labs.com との通信を確立しようとするとします。まず最初に必要な処理は、www.bell-labs.com というドメインネームを数字で表わされるインターネットアドレスへと変換することです。これは複雑な処理で、通常、ローカルあるいはリモートのドメインネームサーバとの通信を必要とします。Styx のモデルでは、この処理は /dev/dns というファイルをオープンし、そのファイルに対して www.bell-labs.com という文字列を書き込むことになります。次にそのまま同じファイルに対して読み込みを行うと、検索の結果が 204.178.16.5 という 12 文字の文字列として返されるはずです。

こうしてインターネットアドレスが得られると、そのアドレスに対する通信路を確立しなくてはなりません。このためには、/net/tcp/clone というファイルを開き、/net/tcp/43 というような、ディレクトリ名をあらわす文字列を読み込みます。このディレクトリは、新規に作成された一本の TCP/IP チャネルをあらわしています。次に、connect 204.178.16.5 というメッセージを、このディレクトリにある /net/tcp/43/ctl ファイルに書き込むことで、このチャネルによる通信路を確立します。これ以降、www.bell-labs.com との通信はすべて、/net/tcp/43/data というファイルを読み書きすることで行えます。

この手法について、目をとめてもらいたい点がいくつかあります。

- すべてのインタフェースはファイルのようにみえ、すでに C や C++、Java といったプログラム言語から利用できるのと同じ I/O 機能を使ってアクセスできます。しかしながら、これらのインタフェースを提供するファイルには、対応してディスク上に存在するデータファイルはありません。そのかわり、これらのファイルはミドルウェアのプログラムによって提供されているものです。
- 基本的に、インタフェースとのやりとりはバイナリによる表現ではなく、できるかぎり表示可能な文字列を使って行うことになっています。これはつまり、インタフェースとのやりとりのシンタックスが、CPU アーキテクチャや言語の細かい部分に依存しないということです。
- ネットワーク通信機能へのインタフェースである /net の例のように、インタフェースが階層ファイルシステムの一部であるため、これを遠く離れた場所にあるリモートマシンへと開放するのは簡単で、ほぼ自動的に実現されます。

特に、Styx の実装は、きちんと制御されたネットワークへのアクセスを、自然な方法で提供するのを助けます。他の多くの組織と同様 Lucent にも、国際的なインターネットへは接続できない内部ネットワークと、内外ネットワークの間に置かれた数個のゲートウェイがあります。ゲートウェイのマシンだけが内外両方のネットワークに接続され、安全と保証のために、そのマシンの上で管理制御が実現されています。Styx モデルの持つ長所を利用することで、内側から外部のインターネットを利用するのがより簡単になります。上で述べた /net 以下のファイル構造がゲートウェイのマシン上で提供されれば、それは内側のマシンからリモートファイルシステムとして使用することができます。内側のマシンは外部へのネットワークインタフェースを参照できますが、その逆はできない一方通行の通信なので、これは安全と言えます。

デバッギングの例

UNIX システムからとりこまれ、さらに一般化された方法 [4] ですが、オペレーティングシステムで実行中のプロセスの状態を取得したり、これを制御したりするのも、同様なやりかたで実現できます。図 2 に示すように、/proc というディレクトリ内には、システム上で実行中のプロセスそれぞれに対応するサブディレクトリが置かれています。サブ

ディレクトリ名は対応するプロセスのプロセス ID になります。

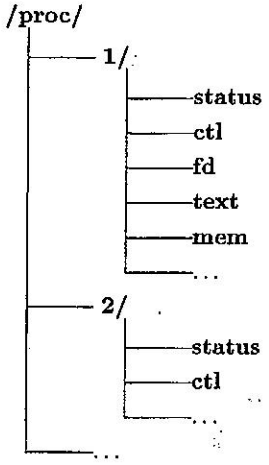


図 2: ファイルシステム上の /proc ディレクトリ

プロセスがもつ様々な要素に対応するファイルが、各プロセスのディレクトリ内に存在します。status ファイルにはプロセスの状態に関する状態が保持されています。ctl に書きこみを行うことで、プロセスの一時停止や実行再開、強制終了などの操作を行うことができます。fd はそのプロセスによって使用されているファイルについて、その名前などを保持しています。text と mem はそれぞれ、プログラムのテキストセグメントやデータセグメントの内容を保持しています。

これらの情報や制御コマンドもまた、できるかぎりテキスト文字列によって表現されています。例として、典型的なプロセスの status ファイルから一行抜き出してみると、次のようになるでしょう。

samterm dmr Read 0 20 2478910 0 0 ...

これらはプログラムの名前、プロセスのオーナと状態、消費した CPU 時間をあらわすいくつかの数字などです。

繰り返しますが、この手法にはいくつかの特長があります。プロセス情報がファイルの形で提供されるため、他のマシン上にある /proc をリモートマウントすることで、即座にリモートデバッギング (他のマシン上のプログラムをデバッグすること) が可能になるということです。マシンに依存しない形で情報を提供することはつまり、デバッグを行っているマシンとリモートマシンが異なる CPU アーキテクチャを持つものだったとしても、だいたいの操作は正しく動作するということになります。プロセスの状態を操作したり、制御をおこなったりするプログラムのほとんどはマシンに依存しないものであり、これらは完全に移植可能なのです (もちろん例外もあります。メモリ上のデータや命令コードをマシンに依存しない形で提供しようとはしていません。)

PathStarTM アクセスサーバの例

Lucent 社製 PathStar アクセスサーバ [12] のデータシェルフは、シェルフに差された回線接続ボードや他のデバイス等を制御コンピュータと接続するの Styx を使用しています。実は、バックプレーンでの上位通信プロトコルが Styx なのです。制御コンピュータによって提供されるファイルシステム階層の一部は、図 3 のようになっています。

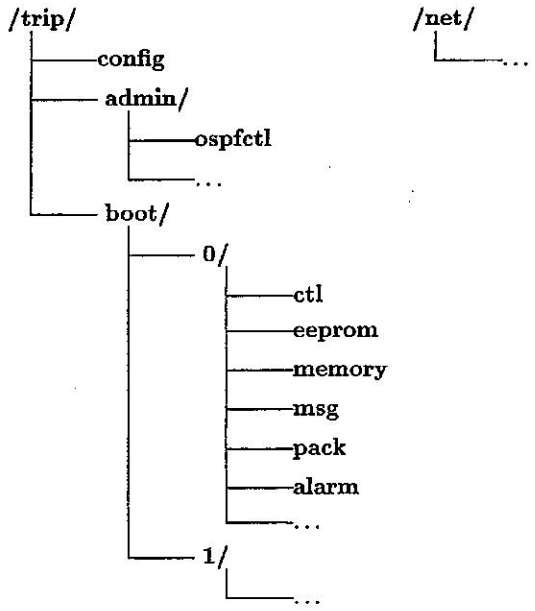


図 3: 制禦コンピュータによって提供されるファイルシステム階層の例

/net 以下のディレクトリは Plan 9 や Inferno と同様、外部 IP ネットワークへのインタフェースとなっています。/trip 以下の階層は、シェルフの制御をおこなうためのものです。

/trip/boot サブディレクトリ以下は、シェルフに差された回線接続ボードや、その他の機器それぞれへのアクセスを提供します。例えば、あるボードを初期化するためには、 reset という文字列をそのボードの ctl ファイルへと書きこみます。一方、制御用のソフトウェアをそのボードの memory ファイルへとコピーし、 reset メッセージを ctl ファイルに書きこむことで、ブートストラッピングが行えます。いったん回線接続ボードが動作しはじめると、他のファイルによってさらに上位のシステムに対するインタフェースが提供されます。 pack は IP パケットをボードと受け渡しするためのポートになります。また、ボードの状態に特に目立ったことがないか調べるため、 alarm から読み込みを行う等々です。

これらの階層はすべて、シェルフ自体からも Styx によって外部に開放されます。外部の構成機器管理ソフトウェア (EMS) は Styx によってシェルフの監視や制御を行います。例えば、EMS は /trip/boot/7/alarm を読むことで、ボードが診断を必要とする状態にあると判断するかもしれません。そして、EMS を実行しているシステムから /trip/boot/7 / 以下にある他のファイルを読み書きすることで、このボードの動作を停止し、診断を行い、おそらくはリセットをかけるか、他のボードに動作を移すといった操作が全ておこなえます。この EMS はネットワークのどこにあってもかまわないわけです。

別の例として、PathStar アクセスサーバでの SNMP の実装例があげられます。 SNMP を動作させている機器は、ネットワーク内の様々な構成要素として分散しているのが普通です。ここで、SNMP の要求を Styx の操作へと変換する、ごく単純な変換プロセスを、ネットワーク内で一つ動作させます。これは実装を非常に単純なものにできるだけでなく、このように自然な形で集約をおこなえる機能があることで、SNMP アクセスを提供するたった一つのプロセスによって、どのように複雑なネットワークサブシステムにも対応することができます。しかしながら、この操作の本質はファイル操作にもとづくため、標準的な認証を行い、信用できるグループだけが SNMP によるアクセスを許可されるよう保証することが簡単に実現できます。ですから、この構造は安全なものと言えます。

ローカルな部分でもまた、このアーキテクチャによって得られる利点があります。それは、下位の具体的な構成機器の詳細と制御部分を切り分ける個所を、設計上の一点に限定することができるということです。これによって、相互の変更を隠蔽することができます。構成機器を柔軟に組み合わせることができるようになりますし、ハードウェア部分に隠れている依存性を気にすることなく、ソフトウェアをアップグレードできます。また、上位の制御ソフトウェアをアップデートすることなく、新しいハードウェアを追加することもできます。

5 セキュリティの問題

故意であれ偶然であれ、システムの動作を阻害するような行為から、システムを守るための機能を Styx は提供します。

下で動作しているファイル通信プロトコルには、サーバが認証に利用するためのユーザとグループを識別する情報が含まれています。例えば、ファイルをオープンするという要求に対して、その要求を発したユーザの ID が、その操作を行なうことが許可されているかどうか、サーバは確認することができます。この機能は汎用目的のオペレーティングシステムに見られるもので、その使い方についてもすでによく知られています。この機能は、パスワードや、さらに安全な方法によって、クライアントの実体が認証される事に依存しています。

Styx がファイルシステムをリモートな資源として、ネットワークを介して開放すために採用した手法は、アプリケーションと、それが利用する資源の両方ともが、認証のためのやりとりをせずにすむよう、資源に対するアクセスを、アプリケーションから透過的に完全に保護する大きな助けとなります。たとえば Inferno では、クライアント-サーバ間での最初の通信確立時に、通信路を監視するための暗号化、あるいはメッセージダイジェストのプロトコルを数種類のうちから自由に選択できるようになっています。サーバ上の資源を利用するアプリケーションは全て妨害から保護され、また、サーバ側は機能が適切な方法で使用されているという保証を確実に得ることができるわけです。以上は一般的なファイルサーバに有効なだけでなく、電話通信業界における安全なリモート管理の実現にも特にあてはまります。

6 まとめ

資源をファイルシステムの一部として提供し、それがリモートファイルシステムとして扱われても構わないというのは、分散システムを構築する際、次のような両極端の間を行く、とても有望な手法です。

1. システム外との通信は全て、構成要素間でメッセージを明示的に送ることで行う。この通信は、ローカルな資源を利用するときとは違った方法でアプリケーションから使用される。
2. 通信は全て、密接に共有された資源を利用して行われる。具体的には CPU からアクセスできる様々なメモリ領域を、大規模なネットワークを介して直接参照できるようにする。アプリケーションは遠隔地のオブジェクトを、自分が実行されている CPU のように、同じマザーボード上にあるかのように読み書きできる。

一つめにあげられているのは、現在のシステムでごく普通に見られるものですが、その機能を利用するオペレーティングシステムやソフトウェアは、大雑把な範囲をカバーするだけです。次にあげられているものを実現するのはさらに難しいです。ネットワークの信頼性は (インターネットのように大きなものは特にそうですが) 特に高いというわけではありませんし、さらに、ネットワーク上にあるマシンでは様々なアーキテクチャのプロセッサやソフトウェアが利用されているからです。

これまで述べられ、主張されてきた設計は、この両極端の間に位置するもので、次のような特長を持っています。

- 簡単で馴染のあるプログラミングモデルによる、名前で指定されるファイルの読み書き。ファイルシステムにはしっかりと定義された名前付け、アクセス、許可の構造があります。

- プラットフォームや言語に無関係である。資源を参照する下位構造はファイルレベルにありますから、どのシステムでもほぼ利用することが可能で、特定の言語やオペレーティングシステムの機能に依存しません。この機能を利用するための、C++ や Java のクラス、C 言語のライブラリを作ることもできます。
- 階層化された名前付けおよびアクセス制御構造。資源の名前付け、アクセスの設計をわかりやすく、良く構造化されたものにする助けになります。
- 簡単な試験とデバッグ作業。細かく規定された、限定されたインタフェースをファイルレベルで利用することで、分散されたもの同士の通信を簡単に観察することができます。
- 低コスト。サポートソフトウェアは数千行のプログラムで作成でき、機器の容量を少し占有するだけですみます。これはクライアントとサーバ両方を合計しての話です。

以上のシステム設計手法は、汎用システムである Plan 9 と Inferno において有効に働き、また電話通信に特化した Mantra[11] や PathStar アクセスサーバを製作するのにも利用されました。この手法は、単一のシステム内での通信と、大規模なディジタルネットワークを構成する異なる機器同士の外部通信両方に、統一された拡張可能な構造を提供します。

参考文献

[1] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and Implementation of the Sun Network File System", Proc. Summer 1985 USENIX Conf., Portland, Oregon, June 1985, pp. 119-130.

[2] Internet RFC 1094.

[3] Plan 9 Programmer's Manual, Second Edition, Vol. 1 and 2, Bell Laboratories, Murray Hill, N.J., 1995.

[4] T. J. Killian, "Processes as Files", Proc. Summer 1984 USENIX Conf., June 1984, Salt Lake City, Utah, June 1984, pp. 203-207.

[5] R. Pike, D.L. Presotto, K. Thompson, H. Trickey, and P. Winterbottom, "The Use of Name Spaces in Plan 9", Op. Sys. Rev., Vol. 27, No. 2, April 1993, pp. 72-76.

[6] D. L. Presotto and P. Winterbottom, "The Organization of Networks in Plan 9", Proc. Winter 1993 USENIX Conf., San Diego, Calif., Jan. 1993, pp. 43-50.

[7] R. Needham, "Names", in Distributed systems, edited by S. Mullender, Addison-Wesley, Reading, Mass., 1989, pp. 89-101.

[8] Paul Leach and Dan Perry, "CIFS: A Common Internet File System", Nov. 1996, http://www.microsoft.com/mind/1196/cifs.htm.

[9] Inferno Programmer's Manual, Third Edition, Vol. 1 and 2, Vita Nuova Holdings Limited, York, England, 2000.

[10] S.M. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. Trickey, and P. Winterbottom, "The Inferno Operating System", Bell Labs Technical Journal Vol. 2, No. 1, Winter 1997.

[11] R. A. Lakshmi-Ratan, "The Lucent Technologies Softswitch-Realizing the Promise of Convergence", Bell Labs Technical Journal, Vol. 4, No. 2, April-June 1999, pp. 174-196.

[12] J. M. Fossaceca, J. D. Sandoz, and P. Winterbottom, "The PathStar Access Server: Facilitating Carrier-Scale Packet Telephony", Bell Labs Technical Journal, Vol. 3, No. 4, October-December 1998, pp. 86-102.

編集後記

xmoe ⟨xmoe@mma.club.uec.ac.jp⟩

印刷まで

正確にあと1時間、なのですが、百萬石担当 (1) の abe さんは、「人生は暗転する一方だ」というメールを残したまま、来てくれません (笑)。 自分の記事も、まだ書き終っていないので、書く必要があります。マスタの印刷をどうすればいいのか分りません。大学の対外線が調子悪いみたいだったので、フロッピーディスクテクノロジで、百萬石 tarball を持って来たのですが、

```
[xmoe@pulse ~]$ tar -xzvf ./100mangoku-2002.Spring.1.tar.gz
...
gzip: stdin: unexpected end of file
tar: Child returned status 1
tar: Error exit delayed from previous errors
[xmoe@pulse ~]$
```

今年度初の危機的状況です。

ということで、この百萬石が、皆さんのてに届くことを願って。誤植が多かったりするかもしれないけど、勘弁してね (はぁと) (xmoe)

2002 spring

百萬石 for fresher edition